The journal for the international Ada community

Ada User Journal

Volume 46
Number 3
September 2025

Editorial 139
Quarterly News Digest 140
Conference Calendar 163
Forthcoming Events 170

Articles from the AEiC 2025 Work-in-Progress Session

D. Pedro et. al. *Automated Execution of Attack Chains to Train Data-Driven IDS in Cyber-physical Systems* 172

M. Samadi et. al. *Task-to-Accelerator Mapping for Heterogeneous Systems Using Heuristics* 176

Articles from the AEiC 2025 Industrial Track

C. Dross et. al. *Reasoning about Subprogram Termination in SPARK* 180

Articles from the Ada Developer Room at FOSDEM 2025

D. Craeynest et. al. *12th Ada Developer Room at FOSDEM 2025* 185
F. Oleo Blanco *Updates on the Ada Ecosystem from the Past Few Years* 186
A. J. Ianozi *Get Started with Ada in 2 Minutes or Less* 188
G. de Montmollin *Advent of Compression: Writing a Working BZip2 Encoder in Adafrom Scratch in a Few Days* 190
G. de Montmollin *Ada and Mini-Ada: a Solution to the Two-Language Problem* 191
F. Oleo Blanco *Understanding Liquid Types, Contracts and Formal Verification with Ada/SPARK* 192
O. Scherer *The State of Rust Trying to Catch Up with Ada* 195
C. Sagaert *Big Integers for Cryptography in SPARK* 196
J. Verschelde *Multiword Arithmetic and Parallel Computing* 198
C. Simon *Developing Device Drivers for Ironclad Using Ada* 200
T. McGlinn *Kids Learn to Code with AdaBots* 202

Produced by Ada-Europe

Editor in Chief

António Casimiro — University of Lisbon, Portugal
AUJ_Editor@Ada-Europe.org

Ada User Journal Editorial Board

Luís Miguel Pinho — Polytechnic Institute of Porto, Portugal
Associate Editor — *lmp@isep.ipp.pt*

Patricia López Martínez — Universidad de Cantabria, Spain
Assistant Editor — *lopezpa@unican.es*

Dirk Craeynest — KU Leuven, Belgium
Events Editor — *Dirk.Craeynest@cs.kuleuven.be*

Alejandro R. Mosteo — Centro Universitario de la Defensa, Zaragoza, Spain
News Editor — *amosteo@unizar.es*

Ada-Europe Board

Luís Miguel Pinho (President) Polytechnic Institute of Porto	Portugal
Dirk Craeynest (Vice-President) Ada-Belgium & KU Leuven	Belgium
Dene Brown (General Secretary) SysAda Limited	United Kingdom
Ahlan Marriott (Treasurer) White Elephant GmbH	Switzerland
António Casimiro (Ada User Journal) University of Lisbon	Portugal

Ada-Europe General Secretary

Dene Brown
SysAda Limited
Signal Business Center
2 Innotec Drive
BT19 7PD Bangor
Northern Ireland, UK

Tel: +44 2891 520 560
Email: Secretary@Ada-Europe.org
URL: www.ada-europe.org

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 46
Number 3
September 2025

Contents

Page

Editorial Policy for Ada User Journal 138

Editorial 139

Quarterly News Digest 140

Conference Calendar 163

Forthcoming Events 170

Articles from the AEiC 2025 Work-in-Progress Session

D. Pedro, J. Cecílio, P. M. Ferreira, A. Oliveira de Sá
"*Automated Execution of Attack Chains to Train Data-Driven IDS in Cyber-physical Systems*" 172

M. Samadi, T. Carvalho, L. M. Pinho, S. Royuela
"*Task-to-Accelerator Mapping for Heterogeneous Systems Using Heuristics*" 176

Articles from the AEiC 2025 Industrial Track

C. Dross, J. Huguet, J. Kanig
"*Reasoning about Subprogram Termination in SPARK*" 180

Articles from the Ada Developer Room at FOSDEM 2025

D. Craeynest, F. Oleo Blanco "*12th Ada Developer Room at FOSDEM 2025*" 185

F. Oleo Blanco "*Updates on the Ada Ecosystem from the Past Few Years*" 186

A. J. Ianozi "*Get Started with Ada in 2 Minutes or Less*" 188

G. de Montmollin "*Advent of Compression: Writing a Working BZip2 Encoder in Ada from Scratch in a Few Days*" 190

G. de Montmollin "*Ada and Mini-Ada: a Solution to the Two-Language Problem*" 191

F. Oleo Blanco "*Understanding Liquid Types, Contracts and Formal Verification with Ada/SPARK*" 192

O. Scherer "*The State of Rust Trying to Catch Up with Ada*" 195

C. Sagaert "*Big Integers for Cryptography in SPARK*" 196

J. Verschelde "*Multiword Arithmetic and Parallel Computing*" 198

C. Simon "*Developing Device Drivers for Ironclad Using Ada*" 200

T. McGlinn "*Kids Learn to Code with AdaBots*" 202

Ada-Europe Associate Members (Ada Organizations) 204

Ada-Europe Sponsors Inside Back Cover

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

In this editorial, my first words are dedicated to thanking Jorge Real for his outstanding work and dedication to the Ada language, Ada-Europe, and the Ada User Journal over the past three decades. Jorge has now retired and, as a result, has stepped down from his role as Deputy Editor of the AUJ. Speaking more specifically about the AUJ and the work we have shared over the last six years, I can only say positive things about Jorge. He has always been there, available, supportive, and ready to offer thoughtful input on editorial matters. Beyond that, he has contributed immensely by reviewing texts, editing the table of contents, preparing the journal covers, and handling the printing and distribution. A remarkable amount of volunteer work, always carried out with professionalism and care. When the COVID pandemic hit, printing moved from Valencia to Lisbon, but Jorge continued to take care of many tasks, including preparing the covers until very recently. Above all, working with Jorge has always been very easy and truly a pleasure, because he is a wonderful person. Thank you so much for everything, Jorge, and enjoy your retirement!

Concerning technical contents, this issue includes papers from three sources. First, we conclude the publication of articles from the AEiC 2025 Work-in-Progress Session, including two articles. The first presents a framework for executing attack chains in simulated CPS environments, enabling the collection of datasets that can be used to improve Intrusion Detection Systems. The second proposes an approach for heterogeneous task-to-accelerator mapping, aiming to improve the performance and predictability of OpenMP applications. Then, this issue includes an article from the AEiC 2025 Industrial Track, authored by Claire Dross, Joffrey Huguet and Johannes Kanig from AdaCore, describing new subprogram contracts designed to enhance the analysis of Ada programs using the SPARK tool. Finally, the issue features a compilation of ten short articles from the 12th Ada Developer Room at FOSDEM 2025. This compilation is introduced by an article prepared by the event organizers, Dirk Craeynest and Fernando Oleo Blanco. All contributions are related to the Ada programming language and summarize the respective presentations delivered at the event.

Last but not least, the issue includes the Quarterly News Digest section, covering the past three months, prepared by Alejandro R. Mosteo. It also includes, as usual, the Conference Calendar and Forthcoming Events sections, prepared by Dirk Craeynest. The latter provides an updated announcement and call for contributions for the 30th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2026), which will take place in Västerås, Sweden, in June 2026.

Antonio Casimiro
Lisboa
September 2025
Email: AUJ_Editor@Ada-Europe.org

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

Preface by the News Editor	140
Ada-related Events	140
Ada-related Resources	142
Ada-related Tools	144
Ada and Operating Systems	149
Ada Inside	149
Ada Practice	151
Ada Frontiers	161
SPARK/Ada	162

[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,

Given the lively discussion in the ada-lang forum, it is becoming difficult to pick just a couple of threads to highlight. I will start with a thread [1] about the benefits of using Ada in the large. It is a bit of preaching to the choir, but interesting nonetheless.

Then I would like to point out some efforts to expand the reach of Ada: a new full toolchain for FreeBSD [2], and two projects related to embedded development: a bare-board runtime generator [3] and configurable runtimes for STM32 [4].

A rather unique announcement has also recently been made by the Indian government: its first 32-bit chip for space applications is being produced, with high-level support for Ada [5]. I could not find exactly what this support entails, so let us keep on the lookout!

Finally, do not miss the various announcements for the Crate of the Year Award, monthly meet-ups, FOSDEM 2026 and AEiC 2026 in the Ada-related Events section.

Sincerely,
Alejandro R. Mosteo

[1] "Experiences with Large Projects", in Ada Practice

[2] "Full GNAT Ada 2022 Toolchain for FreeBSD", in Ada-related Tools

[3] "Bare Board Runtime Generator 0.0.1", in Ada-related Tools

[4] "New Configurable STM32 Bareboard Runtimes", in Ada-related Tools

[5] "Vikram-3201: India's Custom 32-bit Processor Built for Ada", in Ada Inside

Ada-related Events

Info about Ada Monthly Meetup, Ada Developers Workshop 2025 and Some More

From: Irvise
Subject: Info about Ada Monthly Meetup, Ada Developers Workshop 2025 and Some More
Date: Wed, 2 Jul 2025 11:11:32 +0000
Newsgroups: forum.ada-lang.io

I hope you are doing well and that you are getting ready for holidays. I would like to give you all a small update regarding a few things I am involved in and that I know some people have some expectations.

- Ada Monthly Meetup will not be taking place in July nor August. This is due to these months being holidays for many people (including me) and that the Developers Workshop took place a couple of weeks ago. Also, I am quite busy.
- The publishing of the videos for the Ada Developers Workshop are going to take much longer than expected. This is due to my life being extremely busy and things just keep popping up… Sorry for that, but this is just how things are.
- I would also like to apologise again due to the technical issues that took place during the Workshop… Some recordings have had parts of them lost and the people online could not see the webcam feed for most of the day… I should have tested the setup more thoroughly… Either way, it seems there was a bug with OBS and the drivers, so I could not have fixed it, but still…
- The weekend after the Workshop, 20-22 of June, I was in OpenSouthCode [1]. It was my first time there. I gave a 2h long Introduction to Ada in Spanish. Sadly, due to some conditions, not many people attended… Nevertheless, the presentation was recorded. The slides and recording will be made available in the talk's website entry [2]. Sadly, the webcam recording was extremely blurry as I forgot to disable autofocus and the camera did a terrible job during most of the presentation…
- The conference itself was extremely good! I met a ton of people and wonderful communities. I knew there were a lot of cool open source developers in Spain, but I was very impressed by the number and quality of them! It was also great to see the amount of foreigners that attended the conference (50% of the talks were in English). The organisation of the conference was also outstanding and they really focused on networking and people getting together. It was truly lovely and I hope to go back next year.
- I am looking into ways that some things in the Ada community could get some funding… I would like to see the *BSD patches for GNAT upstreamed and improved upon… I would also like to see someone take over supporting GNAT on Apple systems, Zephyr support for Ada (see @jgrivera67 topic in the Workshop), Ada support for RTEMS new build system, GTK 4 support, etc.

Well, that was a bit of talking… Anyhow, I wish you all a great summer!

[1] https://www.opensouthcode.org/conferences/opensouthcode2025

[2] https://www.opensouthcode.org/conferences/opensouthcode2025/program/proposals/851

2025 Crate of the Year Awards

From: Fabien.C
Subject: 2025 Crate of the Year Awards!
Date: Thu, 10 Jul 2025 08:16:09 +0000
Newsgroups: forum.ada-lang.io

Hello here, AdaCore just announced the 4th edition of the Crate of the Year Awards [1]. No need to register, winning crates will be selected from all the crates in the Alire ecosystem.

I am opening this thread as a place for everyone to discuss which crate they think would be a good candidate or even promote their crate.

Have fun and happy hacking!

[1] https://blog.adacore.com/announcing-the-2025-ada-spark-crate-of-the-year-award

Ada Monthly Meetup, 13th of September

From: Irvise
Subject: Ada Monthly Meetup, 13th of September
Date: Wed, 20 Aug 2025 14:17:36 +0000
Newsgroups: forum.ada-lang.io

I would like to announce the September (2025) Ada Monthly Meetup which will be taking place on the 13th of September at 14:00 UTC time (16:00 CEST). As always, the meetup will take place over at Jitsi. The Meetup will also be livestreamed/recorded to YouTube.

Important: due to some personal scheduling issues, this meetup will be taking place the second week of the month, and one hour later than usual. This should be a one-off, but still, be aware.

If someone would like to propose a talk or a topic, feel free to do so! We currently have no proposals. Nevertheless, we will talk about the AEiC conference, a bit of preparation for FOSDEM and then some news, regarding Alire, increase of popularity of Ada in some programming language indexes and other topics that the community may want.

Here are the connection details from previous posts:

The meetup will take place over at Jitsi, a conferencing software that runs on any modern browser. The link is Jitsi Meet [1] The room name is “AdaMonthlyMeetup” and in case it asks for a password, it will be set to “AdaRules”.

I do not want to set up a password, but in case it is needed, it will be the one above without the quotes. The room name is generally not needed as the link should take you directly there, but I want to write it down just in case someone needs it.

Best regards and see you soon!
Fer

P.S: I still have not started video editing the talks of AEiC… I have been very busy… Let's see if I can get some time and energies in the following weeks.

P.S: feel free to repost this in other forums or chats, such as Reddit!

See related post of the same name for more info

[1] https://meet.jit.si/AdaMonthlyMeetup

From: Irvise
Date: Sat, 13 Sep 2025 16:40:54 +0000

The meeting just finished and here are the Minutes of the Meeting:

- @Irvise gave his perspective and feelings about the Ada Developers Workshop which took place during this year’s Ada-Europe international Conference. He has received pretty good feedback and people seemed to be quite happy with it. The recordings of the videos still need to be processed, he apologises for the substantial delay.
- @DirkCraeynest then shared with us the slides that were used during AEiC to present the 2026 edition of the conference! It will be taking place 9-12 June 2026, Västerås, Sweden [1]. The call for papers/presentations in the different tracks is already open. We will try to repeat again the Ada Developers Workshop, as it had quite a lot of attendance.
- @Irvise told us about his experience giving an introductory workshop about Ada. It was given in OpenSouthCode [2]. There was not much attendance but there was good reception. It was an incredible experience and a great conference overall.
- Also, from that meetup, he learnt about SysML v2 [3]. It may be very interesting to see an Ada code generator for it in the future… who knows!
- FOSDEM [4] 2026 has started preparations and we hope to see the official dates and the Call for Devrooms quite soon. Of course, we plan on applying again this year.
- India has developed their own CPU and Ada will be a first class citizen in their tech stack! [5]
- @Irvise pointed out that India, South America and China barely have any representation in the wider Ada community and that they are growing markets that could substantially benefit from it. We may need to have a push or showcase that targets those communities

NVIDIA:

- They published their work on ISO-26262 certification [6] and published a guide on how they did it!
- They certified to ASIL-D their DriveOS, a 7 MLOC OS written in SPARK [7]
- Their White Hat Hackers showcased in Defcon their work on securing over a billion RISC-V chips with the use of Ada! [8] (related article [9]). Slides are here [10] and the live-recording [11]. I sadly cannot find the specific video of their talk, but I know it exists. Please, feel free to link it here.

- Ada keeps on climbing in popularity rankings! [12] We need to keep on sharing the word, teaching and showcasing our passion for it!
- REMINDER: The Ada Crate of the Year Award [13] is still ongoing. It is until the end of the year, but please, don’t leave your projects till the very end!
- @godunko did a PR to get GNAT-LLVM ready for Alire [14] quite a while ago! That is great news, let's see if we can get it accepted!
- Reminder that @JeremyGrosser @kevlar700 and @Irvise are now moderators, so now it is not only Jeremy dealing with the spam. If you have any questions, issues, etc; feel free to contact us or create a forum thread with the category of “site feedback”.
- There are several job postings by Raytheon [15]
- The Ada User Society [16] has improved its website and it will hold the General Assembly on the 7th of October. If you are a member, do not forget to attend!
- @Irvise has started discussions with them about creating a fund to provide prizes, money or incentives for people to support and work on relevant Ada programs and improve the wider Ada experience. Expect more news in the future, but for the time being, this is all I can share.
- Luke shared a method for funding which was used for the Amiga [17]
- @mosteo recently became a father and his work on Alire has been paused for the time being. So no need to panic over fewer commits being done there! And let this serve as a reminder that anybody can contribute to Alire!

EDIT: I forgot to link the video recording/livestream https://www.youtube.com/watch?v=qnir_hk1ZC4

Thank you all that attended!

Best regards,
Fer

[1] https://www.ada-europe.org/conference2026/

[2] https://www.opensouthcode.org/conferences/opensouthcode2025/program/proposals/851

[3] https://www.omg.org/sysml/sysmlv2/

[4] https://fosdem.org/2025/

[5] https://forum.ada-lang.io/t/vikram-3201-india-s-custom-32-bit-processor-built-for-ada/3793

[6] https://forum.ada-lang.io/t/nvidia-publishes-spark-process-to-meet-iso-26262-requirements/2114

[7] https://forum.ada-lang.io/t/article-nvidia-drives-ada-and-spark-into-driverless-cars/2117

[8] https://defcon.org/html/defcon-33/dc-33-speakers.html#content_60337

[9] https://riscv.org/blog/2025/02/how-nvidia-shipped-one-billion-risc-v-cores-in-2024/

[10] https://media.defcon.org/DEF CON 33/DEF CON 33 presentations/Adam Zabrocki Marko Mitic – How to

secure unique ecosystem shipping 1 billion%2B cores.pdf

[11] https://www.youtube.com/live/MEPF0dEAjPg

[12] https://forum.ada-lang.io/t/ada-on-the-pypl-index/425/11

[13] https://forum.ada-lang.io/t/2025-crate-of-the-year-awards/2203

[14] https://github.com/alire-project/GNAT-FSF-builds/pull/83

[15] https://forum.ada-lang.io/t/several-ada-jobs-at-raytheon/3798

[16] https://www.ada-user.org/

[17] https://will-templates.com/amigabounty/

From: DirkCraeynest
Date: Tue, 16 Sep 2025 19:07:05 +0000

> the slides that were used during AEiC to present the 2026 edition of the conference

They were also posted on the AEiC 2026 page in Ada-Belgium's calendar of Conferences and events for the international Ada community [1]. See the invitation to Ada-Europe 2026 at MDU [2] that was presented during the AEiC 2025 closing session.

[1] https://people.cs.kuleuven.be/~dirk.craeynest/ada-belgium/events/list.html

[2] https://people.cs.kuleuven.be/~dirk.craeynest/ada-belgium/events/26/260609-aeic-mdu.pdf

FOSDEM 2026, Call for Participation

From: Irvise
Subject: FOSDEM 2026, Call for Participation
Date: Sun, 21 Sep 2025 20:29:19 +0000
Newsgroups: forum.ada-lang.io

The CfP has been issued! [1]

I am creating a thread quickly to track our submission and submission message

IMPORTANT EDIT: if you would like to participate as a co-organiser, please, please, please, let us know ASAP!!!

[1] https://fosdem.org/2026/news/2025-09-21-call-for-participation/

CfC 30th Ada-Europe Int. Conf. Reliable Software Technologies

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: CfC 30th Ada-Europe Int. Conf. Reliable Software Technologies
Date: Wed, 24 Sep 2025 15:51:11 +0000
Newsgroups: comp.lang.ada

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

Ada-related Resources

[Delta counts are interpolated from July 1st to October 1st. Tabulated raw data is available at https://bit.ly/auj-signals —arm]

Ada on Social Media

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada on Social Media
Date: 13 Oct 2025 23:36 CET
To: Ada User Journal readership

Ada groups on various social media:

- Reddit: 2_680* members [1]
- LinkedIn: 3_704 (+46) members [2]
- Stack Overflow: 2_443 (=) questions [3]
- Ada-lang.io: 404 (+20) users [4]
- Gitter: 293 (+4) people [5]
- Telegram: 226 (-1) users [6]
- Discord: 197 (+5) users [7]
- Libera.Chat: 75 (=) concurrent users [8]

[*] Reddit has changed how it measures engagement. Rather than showing all subscribers since the Epoch, it now shows a running average of active users during the last month. Delta count is therefore not representative of any real changes in membership.

[1] https://old.reddit.com/r/ada/

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

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

[4] https://forum.ada-lang.io/u

[5] https://app.gitter.im/#/room/#ada-lang_Lobby:gitter.im

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

[7] https://discord.gg/fEmGe87c

[8] https://netsplit.de/channels/details.php?room=%23ada&net=Libera.Chat

Repositories of Open Source Software

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Repositories of Open Source software
Date: 13 Oct 2025 23:36 CET
To: Ada User Journal readership

GitHub: >7_800* (=) repositories [1]
>1_000* (=) developers [1]
Alire: 1_432 (+6) releases [2]
585 (+26) crates [3]
Rosetta Code: 1_051 (+25) examples [4]
43 (=) developers [5]
SourceForge: 241 (-1) projects [6]
Open Hub: 214 (=) projects [7]
Codelabs: 64 (+1) repositories [8]
Bitbucket: 36* (-1) repositories [9]

*This number is a lower bound due to site search limitations.

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

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

[3] `alr search --list --full`

[4] https://rosettacode.org/wiki/Category:Ada

[5] https://rosettacode.org/wiki/Category:Ada_User

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

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

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

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

Language Popularity Rankings

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada in language popularity rankings
Date: 13 Oct 2025 23:36 CET
To: Ada User Journal readership

[Positive ranking changes mean to go up in the ranking. —arm]

- PYPL Index: 11 (+4) 2.17% (+0.65%) [2]
- TIOBE Index: 18 (-6) 1.06% (-0.59%) [1]
- Stack Overflow Survey: 40 (=) 0.9% (=) [3]
- IEEE Spectrum (trending): 43 (+3) Score: 0.0038 (+0.0015) [4]
- IEEE Spectrum (general): 46 (+4) Score: 0.0027 (+0.0012) [4]
- IEEE Spectrum (jobs): 40 (+14) Score: 0.0017 (+0.0016) [4]
- Languish Trends: 165 (-11) 0.01% (=) [5]

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

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

[3] https://survey.stackoverflow.co/2024/

[4] https://spectrum.ieee.org/top-programming-languages/

[5] https://tjpalmer.github.io/languish/

Ada on the PYPL Index

From: zertovitch
Subject: Ada on the Pypl Index
Date: Tue, 1 Jul 2025 23:04:51 +0000
Newsgroups: forum.ada-lang.io

1.65%

Through the roof

From: pyj
Date: Wed, 2 Jul 2025 01:09:14 +0000

Needs more up arrows

From: gast
Date: Thu, 3 Jul 2025 21:08:26 +0000

What is the main reason for Ada's growing popularity this year?

From: charlie5
Date: Fri, 4 Jul 2025 00:54:23 +0000

Reached critical mass? …

From: F-Loyer
Date: Sun, 6 Jul 2025 11:53:25 +0000

Programmers usually don't choose (only) a language, but an ecosystem. If I want to program the computation of a filter response and the drawing a bode plot, I guess I would choose Python because of the matplotlib (even if Ada would be nice with its included complex computation). It is also a strength of Java: many things available out of the box (GUI binding, Web, etc.).

I guess Alire will have an important value for many programmers who are considering Ada to develop something. Unfortunately, not every library can be easily installed with Alire.

From: zertovitch
Date: Thu, 4 Sep 2025 15:17:27 +0000

September 2025: 2.11%

From: krischik
Date: Thu, 4 Sep 2025 16:31:45 +0000

Yes, Ada at place 12

Ada almost caught up with Rust

And at TIOBE it's similar, place 13

From: zertovitch
Date: Fri, 5 Sep 2025 00:26:42 +0000

Note, regarding TIOBE, that the keyword "nvidia" is excluded from the Web searches. It is explained in "This Month's Changes in the Index" (August 2025):

> W. H. told us that programming language Ada might have been boosted in the TIOBE index because of the Ada Lovelace architecture of NVIDIA. For this reason we have added "-NVIDIA" to Ada's search term. As a consequence, Ada dropped from position #9 last month to position #13 this month.

This is unfortunate because the exclusion discards lots of articles about NVIDIA's own usage of Ada.

From: DirkCraeynest
Date: Fri, 5 Sep 2025 13:28:13 +0000

Hence, an additional correction of Ada's search term is required. +"NVIDIA Ada programming" maybe?

From: zertovitch
Date: Fri, 5 Sep 2025 18:19:12 +0000

> +"NVIDIA Ada programming" maybe?

With that query, only that exact sequence (theoretically) would be searched, and Ada would land at place +/- #513.

But I have searched a few weeks ago with Google, with a 1-year filter (as the TIOBE guy does) +"ada programming" +"NVIDIA" to see the impact of doing +"ada programming" -"NVIDIA", as the TIOBE guy does currently.

37 hits were shown.

33 hits mentioned the Ada language (so, should be counted).

1 hit mentioned the "Ada Lovelace" GPU (so, should not be counted).

3 were about other topics or some chat in Indonesian (this noise is everywhere, is counted for all programming languages, so there is a mutual neutralization).

So I have suggested to the author just to remove the -"NVIDIA" filter.

From: waleedmebane
Date: Fri, 5 Sep 2025 20:31:19 +0000

How about remove "-NVIDIA" and add '"-Lovelace"?

From: kevlar700
Date: Sat, 6 Sep 2025 00:33:24 +0000

This might be new too and Ada Lovelace is championed in some pro Women establishments of late.

Ada Computer Science

https://adacomputerscience.org/

From: pmnw
Date: Wed, 24 Sep 2025 14:03:17 +0000

Those interested in rankings may find the following interesting.

IEEE Spectrum – 23 Sep 25 [1]

AI Is Redefining the Concept of a Programming Language's Popularity [1]

Python reigns supreme again, but is AI changing the game for programming languages? Find out how coding is transforming.

[1] https://spectrum.ieee.org/top-programming-languages-2025

From: cantanima
Date: Wed, 24 Sep 2025 23:30:32 +0000

> Those interested in rankings may find the following interesting.

TL;DR:

- the good news is that, before long, we won't be worried / squabbling about programming language popularity
- the bad news is that, before long, we'll be worried / squabbling about Large Language Model popularity
- the worse news is that, not much longer after that, people will look at the world of software developers / engineers / etc. much the same way they look at the world of 8-bit cpu hobbyists: "how charmingly quaint!"

NB:

- i said that was the TL;DR, /not/ that it was my personal belief
- i'm being tongue in cheek (hence the)…
- …mostly

Ada Back in the TIOBE Top 20

From: F-Loyer
Subject: Ada Back in the TIOBE Top 20 (march 2025)
Date: Tue, 15 Jul 2025 22:26:56 +0000
Newsgroups: forum.ada-lang.io

In the top 10… (July 2025):

- Python (26,98 %);
- C++ (9,8 %) ;
- C (9,65 %) ;
- Java (8,76 %) ;
- C# (4,87 %) ;
- JavaScript (3,36 %) ;
- Go (Golang) (2, 04%);
- Visual Basic (1,94%);
- Ada (1,77 %);
- Delphi/Object Pascal (1,77%).

Curious Rust is not there.

From: mgrojo
Date: Thu, 31 Jul 2025 18:30:26 +0000

Developer Tech News – 14 Jul 25

Why is 40-year-old programming language Ada hot again? [1]

Ada, a programming language born in the late 70s, has managed to break into the top 10 of the TIOBE Index for July 2025.

Estimated reading time: 4 minutes

[1] https://www.developer-tech.com/news/why-is-40-year-old-programming-language-ada-hot-again/

Old but Cool: Ada Mapping to CORBA

From: Irvise
Subject: Old but Cool: Ada Mapping to CORBA
Date: Sun, 28 Sep 2025 12:39:29 +0000
Newsgroups: forum.ada-lang.io

I recently found that the OMG (Object Management Group) had available the "standard" on how to map Ada to CORBA [1]. It is freely available here for anybody that may be interested About the Ada Language Mapping Specification Version 1.3 [2]

[1] https://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture

[2] https://www.omg.org/spec/ADA

From: charlie5
Date: Mon, 29 Sep 2025 04:24:38 +0000

Hi Fer … Not sure if it's relevant but CORBA is one of the 'personalities' available in PolyORB.

3. Overview of PolyORB personalities — PolyORB 26 User's Guide

https://docs.adacore.com/live/wave/polyorb/html/polyorb_ug/Overview_of_PolyORB_personalities.html#corba

From: cup
Date: Tue, 30 Sep 2025 06:27:27 +0000

If you're translating any Java/C++ CORBA code, the common coding convention was that references to CORBA objects were suffixed _ptr. Just remember that these are CORBA references and have to be deleted the CORBA way: not using the language's native delete methods. This was a very common problem for newbies - caused no end of crashes.

Writing CORBA code in Java was more elegant than C++. Hopefully writing CORBA in Ada isn't as messy as C++.

Ada-related Tools

Bare Board Runtime Generator 0.0.1

From: godunko
Subject: Announce: Bare Board Runtime Generator Version 0.0.1
Date: Sat, 5 Jul 2025 07:39:14 +0000
Newsgroups: forum.ada-lang.io

While Alire provides ready for use runtimes for some boards, it is not trivial to run an application on other boards, and even harder to use Ada tasking. I've developed a runtime generator to create custom runtimes for bare board applications. Runtime can be fine-tuned for a particular application, it is generated near to application's code and does not need to be distributed.

https://github.com/godunko/a0b-tools

I've been able to run a blink led example on STM32F401/411 and STM32G431/474 boards, and on Arduino Due.

From: kevlar700
Date: Sun, 24 Aug 2025 10:37:57 +0000

I would never normally recommend using C however for clock setup and PWR setup then they are probably very well tested by ST so using their SDK code might not be so bad? (caveat: I have shaken my head at their code before, for example their clock setup code uses a global variable that makes it fragile in relation to procedure execution order). It would mean that porting the Ada runtime might be a drop in. I guess it depends if finding the right C #includes are less or more of an issue. Certainly gprbuild makes it easy to do it once you do. The trickier issue is low power parts and making the runtime be compatible with user executed clock changes which is an exercise I shall have to hope to have the time for one day.

From: godunko
Date: Mon, 25 Aug 2025 07:03:59 +0000

Time and tasking support in GNAT RTL should be improved to support flexible CPU frequencies. Right now, CPU frequency is hardcoded.

Do you have experience with changing power modes? For STM changing of CPU clocks configuration requires change of clock prescalers for peripherals. How is it handled?

From: kevlar700
Date: Mon, 25 Aug 2025 10:03:20 +0000

I have only done it with the low power parts L and U which are designed for it but yes some drivers like timers have to get the current speeds and if I recall correctly there are differences in the code generated by STM32Cube IDE. So my Ada code to get the clock speeds had to be adapted between L and U. To save time I currently have bindings to SystemClock_Config, HAL_RCC_GetPCLK1Freq, HAL_RCC_GetSysClockFreq and HAL_RCC_GetHCLKFreq pulling in around 40 .c and .h files. I have a script to compare them to upstream which is not ideal but I guess it's easier to copy in those 40 files from any new chip than compare for changes in the generated code and you can just copy the clock init code (SystemClock_Config) after any changes made in the STM32Cube IDE. The others are in the HAL package with just an example of binding SystemClock_Config as that needs to be added to main once generated based on STs IDEs clock configuration tooling.

From: Lucretia
Date: Sun, 24 Aug 2025 11:58:16 +0000

Can it be used for osdev (not embedded)?

From: godunko
Date: Mon, 25 Aug 2025 07:01:51 +0000

No, sources of the native RTL are not available in bb-runtimes

Why might it be useful?

New Configurable STM32 Bareboard Runtimes

From: damaki
Subject: New Configurable STM32 Bareboard Runtimes
Date: Tue, 8 Jul 2025 10:45:35 +0000
Newsgroups: forum.ada-lang.io

I've recently released some configurable bareboard runtime crates for several STM32 families, with support for the light, light-tasking, and embedded runtime profiles:

STM32F0xx crates [1]:

• light_stm32f0xx [2]

• light_tasking_stm32f0xx [3]

• embedded_stm32f0xx [4]

STM32G0xx crates [5]:

• light_stm32g0xx [6]

• light_tasking_stm32g0xx [7]

• embedded_stm32g0xx [8]

STM32G4xx crates [9]:

• light_stm32g4xx [10]

• light_tasking_stm32g4xx [11]

• embedded_stm32g4xx [12]

The runtimes are based on GNAT FSF 15, but I could do releases for older versions of GNAT if there's a demand for it.

Each runtime supports every device in the family. The runtime is configured for a specific device via crate configuration variables. For example, to configure the light_tasking_stm32g4xx runtime for the STM32G431K6 device, add the following to your project's alire.toml:

```
[configuration.values]
light_tasking_stm32g4xx.
MCU_Sub_Family = "G431"
light_tasking_stm32g4xx.
MCU_Flash_Memory_Size = "6"
```

The clock tree is also configurable. By default the runtime sets up the system clock from the PLL and the high-speed internal (HSI) oscillator, but you can also configure it for the HSE if your board has one (some Nucleo boards seem to have one) or configure a different clock speed if you want. Here's an example of configuring the light_tasking_stm32g4xx runtime to generate a 170 MHz system clock from a 24 MHz HSE oscillator in alire.toml:

```
[configuration.values]

# Configure a 24 MHz HSE crystal oscillator
light_tasking_stm32g4xx.
HSE_Clock_Frequency = 24000000
light_tasking_stm32g4xx.
HSE_Bypass = false

# Select PLLRCLK as the SYSCLK source
light_tasking_stm32g4xx.
SYSCLK_Src = "PLLRCLK"

# Configure the PLL VCO to run at 340 MHz
from the 24 MHz HSE (fVCO = fHSE * (N/M))
light_tasking_stm32g4xx.
PLL_Src = "HSE"
light_tasking_stm32g4xx.
PLL_N_Mul = 85
light_tasking_stm32g4xx.
PLL_M_Div = 6

# Configure the PLLRCLK to run at 170 MHz
from the 340 MHz VCO.
light_tasking_stm32g4xx.PLL_R_Div = 2

# Configure the AHB and APB to also run at
170 MHz
```

```
light_tasking_stm32g4xx.AHB_Pre = "DIV1"
light_tasking_stm32g4xx.APB1_Pre = "DIV1"
light_tasking_stm32g4xx.APB2_Pre = "DIV1"
```

See the projects' GitHub pages (links below) for full details of what is configurable.

The hope is that these runtimes are flexible and easy-to-use enough that they help people get started with a new STM32 project quickly, without needing to build their own runtimes for their specific device/board since they can use and configure these runtimes instead.

[1] https://github.com/damaki/stm32f0xx-runtimes/

[2] https://alire.ada.dev/crates/light_stm32f0xx

[3] https://alire.ada.dev/crates/light_tasking_stm32f0xx

[4] https://alire.ada.dev/crates/embedded_stm32f0xx

[5] https://github.com/damaki/stm32g0xx-runtimes/

[6] https://alire.ada.dev/crates/light_stm32g0xx

[7] https://alire.ada.dev/crates/light_tasking_stm32g0xx

[8] https://alire.ada.dev/crates/embedded_stm32g0xx

[9] https://github.com/damaki/stm32g4xx-runtimes/

[10] https://alire.ada.dev/crates/light_stm32g4xx

[11] https://alire.ada.dev/crates/light_tasking_stm32g4xx

[12] https://alire.ada.dev/crates/embedded_stm32g4xx

Zip-Ada 61

From: zertovitch
Subject: Zip-ada Version 61
Date: Tue, 8 Jul 2025 16:44:39 +0000
Newsgroups: forum.ada-lang.io

Here are blog posts diving into what is "under the hood" in the new BZip2 encoder.

Writing a BZip2 encoder in Ada from scratch in a few days - part 1 [1]

Writing a BZip2 encoder in Ada from scratch in a few days - part 2 [2]

Spoiler: it is also likely already the best BZip2 encoder for a given block size.

[1] https://gautiersblog.blogspot.com/2024/11/writing-bzip2-encoder-in-ada-from.html

[2] https://gautiersblog.blogspot.com/2025/07/writing-bzip2-encoder-in-ada-from.html

Strings Edit 3.9

From: dmitry-kazakov
Subject: Strings Edit 3.9
Date: Sat, 12 Jul 2025 12:56:19 +0000
Newsgroups: forum.ada-lang.io

String processing for Ada [1]

Changes to the previous version:

- Bug fix in Strings_Edit.UTF8.Maps.Is_Prefix;
- Unread was added to Strings_Edit.Streams to return the contents available to read;
- Get function was added to.Strings_Edit.UTF8.Maps.

[1] https://www.dmitry-kazakov.de/ada/strings_edit.htm

Simple Components 4.75

From: dmitry-kazakov
Subject: Simple Components 4.75
Date: Sat, 12 Jul 2025 13:04:41 +0000
Newsgroups: forum.ada-lang.io

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), arbitrary precision arithmetic, pseudo-random non-repeating numbers, symmetric encoding and decoding, IEEE 754 representations support, streams, persistent storage, multiple connections server/client designing tools and protocols implementations.

Simple components for Ada [1]

The new version provides an implementation of SNOBOL-like patterns. The patterns are integrated into the parsing framework. They can be constructed using expressions and then matched against the generic source.

SNOBOL patterns are more powerful than regular expressions. A BNF grammar can be directly translated into pattern. Therefore they can be recursive. Features like immediate assignment and printout are fully supported. Patterns can be extended by user-defined matching functions. Unicode is fully supported. In particular, matching letters involve Unicode categorization.

Changes to the previous version:

- The package Parsers.Generic_Source.Patterns was added to implement SNOBOL-like patterns;
- The package Parsers.Generic_Source.Patterns.Generic_User_Pattern was added to provide user-defined patterns;
- The package Parsers.Generic_Source.Patterns.Generic_Parametrized_User_Pattern was added to provide user-defined patterns with parameter;
- The function Top added to the package Stack_Storage;
- The interface procedure Set_Pointer was modified in the generic package Parsers.Generic_Source.

[1] https://www.dmitry-kazakov.de/ada/components.htm

Ada Version of Curl?

From: zertovitch
Subject: Ada Version of Curl?
Date: Wed, 16 Jul 2025 19:18:15 +0000
Newsgroups: forum.ada-lang.io

I have just read this post: Death by a thousand slops [1]

daniel.haxx.se – 14 Jul 25 [1]

I have previously blogged about the relatively new trend of AI slop in vulnerability reports submitted to curl and how it hurts and exhausts us. This trend does not seem to slow down. [...] Even though the topic is more about fake bugs (as I understand it), the kind of vulnerabilities mentioned (mostly buffer overflows) hint at (or even scream for) a nice project idea: an Ada version of curl:

- Name: Aurel
- Primary API: in Ada
- C API for compatibility and replacing curl

The most possible parts would be made with SPARK. It could be a nice advertisement for Ada/SPARK (my impression is that curl is a very "visible" / pivotal software). But perhaps such a project already exists?

[1] https://daniel.haxx.se/blog/2025/07/14/death-by-a-thousand-slops/

From: OneWingedShark
Date: Wed, 16 Jul 2025 21:29:53 +0000

This is misguided: do /everything/ without respect to C /*first*/… then shim in a C compatibility module for export.

From: dmitry-kazakov
Date: Thu, 17 Jul 2025 08:29:15 +0000

> then shim in a C compatibility module for export.

Well, if the Ada API relies on indefinite types (unbounded arrays, tagged types) and/or generics that would be very difficult. You better plan C API in advance, if you want them.

Furthermore, C will be involved anyway in the form of OpenSSL and/or GNUTLS. I doubt the author plans to reimplement them. Which I would welcome, of course.

To the point. There was an Ada Crypto Library [1] project, which unfortunately is no longer supported.

I studied it a lot with regard to primality and arbitrary precision numbers, when I implemented the Simple Components. It is well written, understandable and contains many encryption algorithms which might be useful. I too implemented ChaCha20 and AEAD and have an Ada ASN.1 implementation (you need it for keys and certificates).

Anyway if anybody is brave (or stupid) enough to give an Ada TLS a try, let it be known!

[1] https://github.com/cforler/Ada-Crypto-Library

From: waleedmebane
Date: Thu, 17 Jul 2025 08:38:48 +0000

How about SparkNaCl?

I have never used it, but the README [1] contains this text:

> 22nd June 2022

> Jon Andrew [2] has kindly contributed implementations of the ChaCha20 stream cipher (in SPARKNaCl.Stream), SHA-256 (in SPARKNaCl.Hashing), SHA-256-based HMAC and HKDF algorithms (in SPARKNaCl.MAC and SPARKNaCl.HKDF respectively) and the an AEAD algorithm (RFC 8439) using ChaCha20 and Poly1305 (in SPARKNaCl.SecretBox), plus additional test cases and clean proofs. These additions take the library well beyond the original specification of NaCl, and move towards supporting TLS 1.3.

[1] https://github.com/rod-chapman/SPARKNaCl/

[2] https://github.com/docandrew

From: docandrew
Date: Fri, 18 Jul 2025 21:54:38 +0000

Yes, it's certainly possible and a verified TLS library in SPARK is a long-term goal (maybe dream) of mine.

SPARKNaCl has all the primitives you'd need, it recently added more SHA functions to make the other TLS 1.3 cipher suites possible.

I have a proof-of-concept for the handshakes here: [1]

I started on my own x509 library [2] but it's pretty rough, Dmitry's might be a better starting point. I think RecordFlux's verified record parsers would also be a really great tool to try to apply to the project.

The building blocks are all there! It just needs a hero with more mental horsepower and/or free time than I have

[1] https://github.com/docandrew/SPARKTLS/blob/fd58359697d4d086738f0e1d2bf058027f7819ff/src/tls.adb

[2] https://github.com/docandrew/SPARKx509

Full GNAT Ada 2022 Toolchain for FreeBSD

From: captain-haddock17
Subject: Full Gnat Ada 2022 Toolchain for Freebsd
Date: Wed, 16 Jul 2025 20:54:47 +0000
Newsgroups: forum.ada-lang.io

I'm pleased to announce the availability of the full GNAT Ada 2022 toolchain for FreeBSD.

- GNAT latest Ada commits on 2025-07-04, with GCC 13, 14, 15.1.1 and 16-devel
- GPRBUILD, latest commits on 2025-03-12
- Alire, 2.1.0 from branch

All the binaries are on AdaForge's GitLab [1] in their "Package registry".

[...]

ACATS-4.2.1 [2]

- is on its way (68 Class L programs built)

Side Note:

There is already a first port of gnat13 done by FreeBSD GCC port maintainer Thierry with whom I had a nice chat former friday, but as I had some issues to build it on my rig, and already had a working gnat12 built mid-2022, I took the challenge to set-up a full CI-CD for our Ada toolchain on our FreeBSD server with build system poudriere.

Next step: PR to FreeBSD maintainer to have it direct in the FreeBSD Port & Pkg eco-system, ready to be downloaded.

William J. F.

[1] https://gitlab.com/adaforge

[2] https://gitlab.com/adaforge/devtools/ada-compilers/Ada_AcaaACATS

Added Gentoo Support to Alire

From: Lucretia
Subject: Added Gentoo Support to Alire
Date: Thu, 31 Jul 2025 22:03:19 +0000
Newsgroups: forum.ada-lang.io

I have a branch [1] which adds in support for portage and emerge into Alire. I cannot debug this as it raises an exception reading the settings TOML on a newline, 0x0a. I recreated my settings from scratch with alr and it still happens.

Also, when you ctrl-c the emerge I get an unhandled exception [2] even though all are handled inside the Install subprogram like all the other package managers.

So, if anyone on gentoo wants to try it out, you'll need to modify an external package in the following way [3].

[1] https://github.com/Lucretia/alire/tree/add-gentoo

[2] https://bpa.st/EB6Q

[3] https://github.com/alire-project/alire-index/compare/stable-1.4.0...Lucretia:alire-index:add-gentoo-sdl2-pkgs

ALI_Parse 1.0

From: zertovitch
Subject: Ali_Parse, V1.0
Date: Fri, 1 Aug 2025 08:34:01 +0000
Newsgroups: forum.ada-lang.io

ALI_Parse is a parser for the .ali files generated by the GNAT Ada compiler.

Alire crate:
https://alire.ada.dev/crates/ali_parse

Project site 1:
https://github.com/zertovitch/ali_parse/

Project site 2:
https://sourceforge.net/projects/ali-parse/

ALI means Ada Library Information. You find the .ali files in the same directories as the object files (.o) containing the machine code produced by GNAT.

Currently, ALI_Parse is focused at cross-references within a set of Ada source files. Two command-line tools using the parser are provided:

- GNATHTML, which generates a set of Web pages from Ada sources
- ALI_Stats, which shows the list of entities, the list of cross-references, the count of references to each entity and the list of files.

ALI_Parse is pure Ada 2012 and doesn't depend on any other resource.

From: zertovitch
Date: Sat, 16 Aug 2025 14:43:19 +0000

An example of a Web view generated by ALI_Parse with the Zip-Ada sources:

https://unzip-ada.sourceforge.io/za_html/index.html

From: Blady
Date: Sun, 17 Aug 2025 15:28:46 +0000

Thanks for reviving GNAThtml

Game Made in Ada

From: SabeDoesThings
Subject: Game Made in Ada
Date: Sun, 3 Aug 2025 22:09:23 +0000
Newsgroups: forum.ada-lang.io

I just made a game in Ada for the GMTK [1] game jam! It's probably not the greatest game ever but it was the first thing I actually really worked on aside from little things. I thought this was a great way to test my knowledge from learning the language and I had a great time with it. If you wanna check it out and maybe rate it for the game jam you are more than welcome. Though I've only checked it on Windows so I don't know how or if it will work on other systems so be aware of that.

Loop of Life by SabeDoesThings

https://sabedoesthings.itch.io/loop-of-life

[1] https://itch.io/jam/gmtk-2025

From: mgrojo
Date: Mon, 4 Aug 2025 10:35:11 +0000

We like to see the Ada source code. I've found it here: https://github.com/SabeDoesThings/Loop-of-Life-src

I tried to compile it using the sdlada crate, but it seems to require another binding or another version of the same binding:

```
loop_of_life.adb:4:06: error: file "sdl-mixer.ads" not found
```

From: SabeDoesThings
Date: Mon, 4 Aug 2025 11:51:23 +0000

yeah sdl-mixer is a separate binding so you would have to add it. So that could be the problem.

https://github.com/ada-game-framework/sdlada_mixer

Simple Raycaster in Ada

From: SabeDoesThings
Subject: Simple Raycaster in Ada
Date: Thu, 7 Aug 2025 12:53:08 +0000
Newsgroups: forum.ada-lang.io

Made a simple raycaster in sdlada based off this article [1] took about an hour or so. The source code and footage can be found here [2].

[1] https://lodev.org/cgtutor/raycasting.html

[2] https://github.com/SabeDoesThings/Simple_Ada_Raycaster

Simple Components 4.76

From: dmitry-kazakov
Subject: Simple Components 4.76
Date: Sun, 10 Aug 2025 13:49:47 +0000
Newsgroups: forum.ada-lang.io

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), arbitrary precision arithmetic, pseudo-random non-repeating numbers, symmetric encoding and decoding, IEEE 754 representations support, streams, persistent storage, multiple connections server/client designing tools and protocols implementations.

Simple components for Ada [1]

Changes to the previous version:

- The parsing example was upgraded from Ada 95 to Ada 2022. The changes include:
- Unicode identifiers;
- new () syntax;
- Container aggregate;
- for expressions in container aggregate;
- declare expressions;
- delta aggregates;
- if expressions;
- case expressions;
- raise expressions;
- The package Parsers.Generic_Ada_Parser.Generic_Dot was added for writing Ada syntax trees in the DOT format of Graphviz;
- The package Parsers.Generic_Ada_Parser.Generic_Text_IO was added for printing Ada syntax trees;
- The package Parsers.Generic_Source.UTF8_Keywords was added.

[...]

[1] https://www.dmitry-kazakov.de/ada/components.htm

GNAT Studio 26.0 for macOS Ventura

From: Blady
Subject: Gnat Studio 26.0 for Macos Ventura
Date: Thu, 14 Aug 2025 19:29:17 +0000
Newsgroups: forum.ada-lang.io

Here is a very preliminary version of GNAT Studio 26.0wa [1] as a standalone app for macOS.

Introduced for the first time last year, the GNATStudio launcher looks for a gnatstudio_launcher.rc file in .gnatstudio folder from either $HOME or $GNATSTUDIO_HOME locations. If it exists, we can define some environment variables with the standard syntax VAR=VALUE. If the VAR exists then VALUE is appended to it. If not, VAR is created with VALUE. Thus, it permits to set extra PATH to GNAT compiler and builder folders or GPR_PROJECT_PATH. If a line begins with '#' then it is not considered. An example file of gnatstudio_launcher.rc is provided in the archive. Modify the content and put it in your .gnatstudio folder.

See readme [1] for details.

Limitation: Ada Language Server has some latencies and doesn't respond when parsing source code with more than 1000 lines. It may be due to some compilation options I missed.

There could be some other limitations that you might meet.

Feel free to report them here.

Any help will be really appreciated to fix these limitations.

[1] https://sourceforge.net/projects/gnuada/files/GNAT_GPL%2520Mac%2520OS%2520X/2025-ventura/

Ahven 2.9

From: Tero Koskinen <tero.koskinen@iki.fi>
Subject: ANN: Ahven 2.9
Date: Mon, 1 Sep 2025 18:32:46 +0000
Newsgroups: comp.lang.ada

Ahven is a simple unit test library for Ada programming language.

Release 2.9 is long overdue maintenance release with following notable changes:

- Alire support
- SPDX identifiers added to all source code files
- Support for custom Set_Up and Tear_Down procedures in test suites

Direct links to the source code distributions:

https://www.ahven-framework.com/releases/ahven-2.9.tar.gz

https://www.ahven-framework.com/releases/ahven-2.9.zip

[...]

Ahven homepage: https://www.ahven-framework.com/

Mercurial source control: https://hg.sr.ht/~tkoskine/ahven

Git mirror: https://git.sr.ht/~tkoskine/ahven/

AdaCL 6.3.0 Release Announcement

From: krischik
Subject: Adacl 6.3.0 Release Announcement
Date: Thu, 4 Sep 2025 12:19:17 +0000
Newsgroups: forum.ada-lang.io

Excited to share the AdaCL 6.3.0 release! This suite of Ada 2022 libraries brings robust tools for app development with wide character support. Check the crates below:

adacl

- Offers strings, tracing, AUnit, smart pointers, and Getopt with wide character support for robust apps.
- Explore: https://adacl.sourceforge.net/adacl/

adacl_eastrings

- Developed by Björn Persson, provides encoding-aware strings for i18n, supporting UCS-4, UTF-8, and more.
- Ideal for multilingual apps. See: https://adacl.sourceforge.net/adacl_eastrings/

adacl_regexp

- Features Ada 2022 regex and SPITBOL patterns with wide char support, enhanced from GNAT.
- Perfect for pattern matching. Check: https://adacl.sourceforge.net/adacl_regexp/adacl_sar

- A powerful search/replace library by Martin Krischik, with wide/wide-wide support for text pipelines.
- View: https://adacl.sourceforge.net/adacl_sar/

hp41cx_tools 1.7.0

From: krischik
Subject: Hp41cx_tool 1.7.0 Release Announcement
Date: Sat, 6 Sep 2025 18:22:50 +0000
Newsgroups: forum.ada-lang.io

hp41cx_tools: Showcasing Ada 2022 and AdaCL with Wide_Character Power

New release of hp41cx_tools crate, a cross-platform suite for decoding and converting HP-41CX emulator FOCAL code, demonstrating the strength of *Ada 2022* and the AdaCL library. Available on Alire [1], this project runs on macOS, Linux, and Windows, and is a stellar showcase for AdaCL's Wide_Character support.

Why It's a Great Ada Showcase

- AdaCL with Wide_Character: Leverages AdaCL.Wide_Strings.Regexp and AdaCL.Wide_Strings.Spitbol.Patterns for context-sensitive conversions of special characters (e.g., Σ, ↑, ≠) in FOCAL code. The adacl-regex crate's Wide_Character support ensures precise handling of Unicode, surpassing standard regex for ALPHA strings and commands.
- Robust AUnit Testing: Comprehensive unit tests, executed via make test, ensure rock-solid reliability, covering decoding and conversion edge cases.
- Ada 2022 Power: Built with modern Ada features for maintainable, high-performance code, handling .px41 and .dm41 memory dumps and conversions to .utf8.focal, .dm41.focal, and .px41.focal formats.

Features

- Decode: Transform HP-41CX emulator hex dumps into FOCAL source code with automatic input detection.
- Convert: Seamlessly convert between Unicode, DM41, and PX-41CX FOCAL formats, handling quote styles and XROM key assignments.
- Pipe-Friendly: Four converters use stdin/stdout for scripting ease.

Note: Encoding is planned for a future release.

[1] https://alire.ada.dev/crates/hp41cx_tools

[2] https://sourceforge.net/p/calculator-scripts/code/ci/master/tree/Tools/hp41cx_tools/src/

Demo of 'arcana' Game

From: charlie5
Subject: Demo of 'arcana' Game Written Ada
Date: Thu, 11 Sep 2025 02:29:44 +0000
Newsgroups: forum.ada-lang.io

Here is a video of a hobby game I've been playing with lately.

The game is written in Ada (of course) and is very much a WIP. The game can be built in single player mode when built with gprbuild or in multi-player mode when built with PolyORB's po_gnatdist, using exactly the same code base. It's only 2d and there is much to do.

Just thought it may be of some interest.

From: mgrojo
Date: Thu, 11 Sep 2025 18:13:30 +0000

Nice! We like source code too. I think it's here: https://codeberg.org/charlie5/arcana

From: charlie5
Date: Thu, 11 Sep 2025 20:22:55 +0000

Yes, that's the arcana git repo.

The game engine library (GEL) is here: https://github.com/charlie5/lace

Tbh, I'm not overly proud of a lot of the code (in both). I overuse access types and there are a lot of TODO's. Frankly the whole thing could do with a re-write, if I had the time.

The game engine also handles 3d: https://www.youtube.com/watch?v=f2Ot8jFjuDw

AWS End of Life

From: heharkon
Subject: Aws End of Life
Date: Tue, 23 Sep 2025 07:28:46 +0000
Newsgroups: forum.ada-lang.io

So ummm… what does this mean in practice?

> AWS is a deprecated product. It will be baselined with the GNAT Pro release 28. After this release, there will be no new versions of this product. Contact your sales representative or send a message to sales@adacore.com to get recommendations for replacements.

From: Heziode
Date: Tue, 23 Sep 2025 13:13:10 +0000

Hum… they will soon sell their families in GNAT Pro…

AdaCore and the Apple Syndrome. Nightmare.

From: AJ-Ianozi
Date: Tue, 23 Sep 2025 14:43:50 +0000

As someone who just recently built a free html5 multiplayer online game using websockets and AWS… and is currently working on an MMORPG that leverages AWS as its login server… oh crap!

Also YASS [1] is using AWS for its templating engine, and YASS is gpl3. Any other open source projects relying on AWS?

EDIT: So if AWS is no longer supported, this may mean the community needs to step up and take over the project. Anyone interested in helping?

[1] https://github.com/yet-another-static-site-generator/yass

From: Heziode
Date: Tue, 23 Sep 2025 18:14:41 +0000

Ping @stcarrez, I think you will also be impacted with AWA [1]

[1] https://github.com/stcarrez/ada-awa/tree/master

From: stcarrez
Date: Tue, 23 Sep 2025 21:33:25 +0000

Yes, I was aware since Fabien talked about it offline at Ada Europe in June.

AWA [1] is using AWS but can also be based on EWS [2]. I also have a prototype server based on Simple Components [3] from Dimitry but it has some issues.

AWS is still a good choice for me as long as it is open source.

The question will be whether AdaCore allows the open source community to take over the project and continue if some developers are interested. Personally, I already have some specific branches that solve build issues on NetBSD or FreeBSD (by getting rid of GNATColl dependency which brings lots of complexity and build issues).

[1] https://github.com/stcarrez/ada-awa

[2] https://github.com/simonjwright/ews

[3] https://dmitry-kazakov.de/ada/components.htm

From: stcarrez
Date: Wed, 24 Sep 2025 21:30:25 +0000

I've seen some companies make changes on the license of projects over time. The copyright holder is AdaCore (if I'm not wrong) which means that they can change the license at any time. [...]

If AdaCore transfers the copyright to some Foundation (Apache?), or some non-profit organization (Ada User Society?) this creates a more positive path for the future.

From: robdaemon
Date: Wed, 24 Sep 2025 21:47:50 +0000

"Take over" isn't quite the right way to look at it. This can take a few forms:

- Asking AdaCore to put the repository under community ownership. It would likely have to move GitHub organizations, and needs maintainers.
- Anyone can fork it, since it's GPL, but you can't call it AWS anymore.

Those are the two main ways of "taking over", but the latter removes the AWS name, since AdaCore would retain the copyright.

Ada and Operating Systems

Ada.Text_IO with -mwindows Switch (MSYS2)

From: jere
Subject: Ada.text_io Paired with Mwindows Switch (msys2)
Date: Fri, 1 Aug 2025 14:54:55 +0000
Newsgroups: forum.ada-lang.io

Using the -mwindows switch with an application that uses Ada.Text_IO seemingly causes the application to either exit / exception out. I was wondering if anyone knew of another switch that I could pair with this to allow the program to run. The text_io stuff is just for debug, so I only specify the -mwindows switch in release mode, but I do want it to spawn the console and print stuff in debug mode.

From: godunko
Date: Fri, 1 Aug 2025 17:12:56 +0000

You can use VSS.Text_Streams.Standards.Output_Stream instead, it outputs to stdout when it is available and does nothing otherwise.

From: jere
Date: Fri, 1 Aug 2025 18:21:59 +0000

Thanks. That's kind of the way I am currently going. I can't get VSS, but I'm emulating the same thing using pragma Debug and wrapping that in a function for now.

Ada Inside

Vikram-3201: India's Custom 32-bit Processor Built for Ada

From: dragon-spirit-wtp
Subject: Vikram-3201: India's Custom 32-bit Processor Built for Ada
Date: Wed, 3 Sep 2025 06:36:59 +0000
Newsgroups: forum.ada-lang.io

India's first 32-bit microprocessor, Vikram-3201, was entirely designed and developed by the country. It has its own custom instruction set and was specifically designed to work seamlessly with the Ada programming language, due to being in widespread use in mission-critical systems, particularly in aerospace and defence applications. Even a custom compiler and other tools were developed for it.

"What Is Vikram-3201 Chip? India's Leap In Indigenous Semiconductor Technology | Indian Defence News"

https://www.indiandefensenews.in/2025/09/what-is-vikram-3201-chip-indias-leap-in.html?m=1

From: JeremyGrosser
Date: Wed, 3 Sep 2025 09:36:02 +0000

ISRO & SCL develops 32-bit Microprocessors for space applications [1]

https://scl.gov.in/products_pdf/SC1130-0_Information_sheet.pdf

It's a 32-bit extension of the 16-bit VIKRAM1601. Both the 16 and 32 bit chips are fabbed on 180nm.

[1] https://www.isro.gov.in/vikram3201.html

Ada for Xtensa ESP32 (LX6/LX7) [draft]

From: alexispaez
Subject: Ada for Xtensa Esp32 (lx6/lx7) [draft!]
Date: Fri, 19 Sep 2025 09:23:19 +0000
Newsgroups: forum.ada-lang.io

Will this work with an ESP32-WROOM-32? I just saw it on sale at AZ-Delivery

From: Irvise
Date: Fri, 19 Sep 2025 15:31:27 +0000

ESPRESSIF does not recommend it for new projects https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

It is a rather old CPU by now. However, it has an Xtensa CPU, so it should work with the toolchain. It is probably under discount in order to get rid of extra inventory.

Newer CPUs/Boards include the ESP32-C6, C3, S3, etc

From: Max
Date: Fri, 19 Sep 2025 19:59:14 +0000

It should work in theory, but right now nobody ports Ada runtime for esp32. We only have a compiler.

From: alexispaez
Date: Mon, 22 Sep 2025 08:09:38 +0000

Please excuse my ignorance (I'm trying to learn) …

What does not have an available runtime mean and does it apply to all ESP32 boards?

Does it mean you can't use any Ada library code and you need to do programming on pure bare metal?

What is required to port an Ada runtime for ESP32? Just wondering if it's something I could take on, for didactic reasons, or if it's totally over my head for now

From: Max
Date: Mon, 22 Sep 2025 16:36:58 +0000

> What does not have an available runtime mean and does it apply to all ESP32 boards?

It means that our toolchain for ESP32 is incomplete. We have only a compiler, but not runtime (for any board). So with the compiler one could starting porting Ada runtime and build it: get libgnat.a and a bunch of *.ali files from runtime source files. Then having a compiler and runtime anyone could build and link an Ada application.

> Does it mean you can't use any Ada library code and you need to do programming on pure bare metal?

Without runtime you can't compile any user's compilation unit (package/subprogram), because it depends on the Standard package that is part of runtime. During compilation the compiler reads the System package to find target properties. In turn, runtime comes in several types. The light runtime is most thin and most easy to port. It provides only basic support, you won't be able to use tasks, protected objects, but you can build an application. light-tasking runtime has support for tasks/protected objects with some restrictions (See Ravenscar profile). To get full Ada you need full Ada runtime, that is very big and probably requires real OS underneath.

> What is required to port an Ada runtime for ESP32? Just wondering if it's something I could take on, for didactic reasons, or if it's totally over my head for now

Porting runtime could be interesting and you could find out much about how it works. From my point of view it is not hard, at least for light runtime. You will need to find out

- compiler flags for your particular chip
- flash layout to create a linker script
- number and names of interrupts to write interrupt vector (in assembly)
- create a starter assembly chunk to initialize stack, clean bss memory section

Some links:

- GitHub - Fabien-Chouteau/bare_runtime: Minimal Ada/SPARK run-time for embedded or other restricted targets [1]
- 5. Predefined GNAT Pro Run-Times — GNAT User's Guide Supplement for Cross Platforms 26.0w documentation [2]
- Embedded Systems | ada-lang.io, an Ada community site [3]
- GitHub - AdaCore/startup-gen: A startup code generator for embedded projects [4]
- Startup-gen User's Guide — startup-gen documentation [5]

Don't hesitate to ask questions!

[1] https://github.com/Fabien-Chouteau/bare_runtime

[2] https://docs.adacore.com/gnat_ugx-docs/html/gnat_ugx/gnat_ugx/gnat_runtimes.html

[3] https://ada-lang.io/docs/learn/getting-started/embedded

[4] https://github.com/AdaCore/startup-gen

[5] https://docs.adacore.com/live/wave/startup-gen/html/startup_gen_ug/index.html

From: Irvise
Date: Mon, 22 Sep 2025 19:39:41 +0000

Adding info to what Maxim just linked, the AdaCore blog has quite a few posts about this topic:

> Porting the Ada Runtime to a new ARM board [1]

> by Jérôme Lambourg – Sep 03, 2018. A step by step tutorial to adapt the ARM runtime to new MCUs/boards.

> Ada on FPGAs with PicoRV32 [2]

> by Fabien Chouteau – Oct 08, 2018. When I bought the TinyFPGA-BX board, I thought it would be an opportunity to play a little bit with an FPGA, learn some Verilog or VHDL. But when I discovered that it was possible to have a RISC-V CPU on it, I knew I… [...]

> Starting micro-controller Ada drivers in the Alire ecosystem [3]

> Open-Source Ada: From Gateware to Application [4]

> by Olivier Henley – Feb 10, 2025. The Neorv32 BIOS project demonstrates how Ada can serve as a powerful alternative to C in open-source embedded development.

[1] https://blog.adacore.com/porting-the-ada-runtime-to-a-new-arm-board

[2] https://blog.adacore.com/ada-on-fpgas-with-picorv32

[3] https://blog.adacore.com/starting-micro-controller-ada-drivers-in-the-alire-ecosystem

[4] https://blog.adacore.com/open-source-ada-from-gateware-to-application

Using Both Cores of RP2040

From: RREE
Subject: Using Both Cores of Rp2040
Date: Sun, 21 Sep 2025 20:13:28 +0000
Newsgroups: forum.ada-lang.io

I am trying to use both cores of a RPi Pico RP2040. Unfortunately the example multicore program in Pico_Examples does not blink. It gets stuck when starting a procedure in the second core. When I stop the program in GDB, it stops in RP.Multicore.Launch_Core1.

```
Starting program: /home/re/Devel/rp2040/pico_examples_2.2.0_ba306c6f/multicore/obj/main
^C
Program received signal SIGINT, Interrupt.
0x10003d3e in rp.multicore.launch_core1 ()
(gdb)
```

I am now on the way to use a tasking runtime. That was discussed before here [1]. Unfortunately I cannot find an example program (simple or not). Do I have to declare a task for core0? If not, how would I communicate between the two tasks? Can I still use the internal FIFO queue or is that excluded in the tasking runtime?

[1] https://forum.ada-lang.io/t/ravenscar-profile-tasking-on-rp2040/744

From: Fabien.C
Date: Mon, 22 Sep 2025 08:51:14 +0000

Which run-time are you using?

From: RREE
Date: Mon, 22 Sep 2025 11:23:43 +0000

The original problem (program stuck in launch_core1) arises in the unmodified pico_examples crate. It uses light-cortex-m0p for all examples, including the multicore. I tried with gcc-14 and gcc-15.

The questions in the last paragraph were a preparation for switching to light-tasking-rpi-pico (or light-tasking-rpi-pico-smp). Perhaps that works better, but I am missing some examples. So I haven't tried yet.

From: damaki
Date: Mon, 22 Sep 2025 13:10:47 +0000

I don't have an example to hand right now for using the tasking runtime on the RP2040, but to answer your questions about tasking:

> Do I have to declare a task for core0?

It's not mandatory as the environment task (which calls your program's main) runs on core0. You can also create multiple tasks on core0 if you wish; the runtime will use priority-based scheduling to manage task execution on the same core.

> how would I communicate between the two tasks?

You can use standard Ada tasking mechanisms for inter-task communication, i.e. protected objects [1].

> Can I still use the internal FIFO queue or is that excluded in the tasking runtime?

The SIO FIFO is reserved for use by the runtime only (the runtime uses this to "poke" the other core to wake it up when it might need to (re)schedule a task).

Note that if you want to use both cores with the tasking runtime then you'll need to use the light-tasking-rpi-pico-smp runtime (the other runtime only runs on core0). To declare a task that runs on the second core you can declare the task at package level like this:

```
task My_Task with CPU => 2;
```

[1] https://learn.adacore.com/courses/intro-to-ada/chapters/tasking.html#protected-objects

From: damaki
Date: Thu, 25 Sep 2025 07:53:35 +0000

It looks like both rp2040_hal and the tasking runtime are both trying to use the same alarm interrupt (ALARM2). The runtime reserves this peripheral and interrupt to implement the semantics of Ada's delay and delay until statement. The last chance handler is being reached because RP.Timer.Interrupts also tries to register an interrupt handler for ALARM2, which is rejected by the runtime because that is reserved.

As a workaround I recommend avoiding the use of RP.Timer.Interrupts (don't with it at all in your code) and use Ada's delay and delay until statements instead when using the tasking runtime.

The lack of documentation/examples about the tasking runtimes is certainly something that could be improved.

From: RREE
Date: Wed, 24 Sep 2025 18:22:10 +0000

OK, something strange. It works and the program runs fine if I remove the debugger and cycle power. It does NOT blink if the debugger is attached or if power was not cycled (i.e. debugger removed but power left on)

From: damaki
Date: Thu, 25 Sep 2025 07:58:13 +0000

> OK, something strange. It works and the program runs fine if I remove the debugger and cycle power. It does NOT blink if the debugger is attached or if power was not cycled (i.e. debugger removed but power left on)

I notice that rp2040_hal doesn't provide a way to reset core1 before launching it. I wonder if that could be contributing to the problem.

In the tasking runtimes, I made it so that it hard-resets core1 before launching it to ensure it is in a known state (i.e. the core1 bootrom is waiting for the handshake protocol).

You could try copying this code to reset core1 before launching it and see if that helps:

https://github.com/AdaCore/bb-runtimes/blob/2ebde6e89a7fe60f38faa4af4671278f59ae6e86/src/s-bbsumu__rp2040.adb#L204-L219

From: RREE
Date: Thu, 25 Sep 2025 19:37:29 +0000

Thanks very much. Adding your reset routine actually solved the problem. Now the code starts on both cores, even without detaching the debugger and without power cycling. I only had to replace the 0 and 1 in your code by False and True. I created a corresponding PR for rp2040_hal.

Ada Practice

A Gem Amongst Gems

From: Fabien.C
Subject: A Gem Amongst Gems
Date: Fri, 4 Jul 2025 13:38:24 +0000
Newsgroups: forum.ada-lang.io

In the process of reviewing the Ada Gems, I was amazed by this feature:

Gem #20: Using pragma Shared_Passive for data persistence

https://www.adacore.com/gems/ada-gem-20

[This gem describes how a pragma Shared_Passive in a package makes global variables persistent across program runs, with the help of an auxiliary file that is transparently written and read by the runtime. –arm]

Who knew about this one? (Honestly)

From: dmitry-kazakov
Date: Fri, 4 Jul 2025 14:27:37 +0000

I did not know it, though I worked on persistence a lot.

However there are issues with that:

- In practice you want to use some other backend than arbitrary files, typically a database or a configuration file in a certain format like registry or XML;
- Portability is a big issue. You want to access storage from different platforms and move it across the platforms;
- You need a lot of work for linked structures, like graphs, links etc;
- You also need to handle tagged types, i.e. you need some means to store and restore tags and then construct objects from;
- External references are another problem. You might have not only internal links that can be stored as offsets to the first allocated element, but also a closure of objects your persistent object refers to. Since objects can be shared you need to handle multiple references and maintain a catalogue of objects in the persistent storage. Then you need some policy of updating shared objects.

From: jere
Date: Fri, 4 Jul 2025 18:06:40 +0000

I really like the Ada gems series for things just like this. I hope they never take the archive down as it is a really valuable resource.

From: OneWingedShark
Date: Fri, 4 Jul 2025 18:54:58 +0000

> Who knew about this one?

I did.

I used it to implement configuration for a bit of custom logging. — Basically a store of Pascal-style strings (256-bytes: 1 length byte, followed by the string data), plus the integral data.

> In practice you want to use some other backend than arbitrary files

I don't know that I agree; for many things, yes… but for small things with very few parameters?

> Portability is a big issue. You want to access storage from different platforms and move it across the platforms;

I mean, you're using Shared_Passive, which means that you should be able to hook it into a DSA program easily. (Though, TBF, I think the LRM stance on DSA programs are that it's implementation-defined; which makes sense: incorporating a truly portable RPC/data-communication protocol would be (a) constraining all DSA programs to those protocols, and (b) would entail incorporation-by-reference to something that would likely be ASN.1 or something of similar capability.)

> You need a lot of work for linked structures, like graphs, links etc

This is perhaps the place that current Ada disappoints the most: it's rather obvious that the original (Ada83) design was done with the idea of being able to incorporate self-referential data structures like the aforementioned, when "compiler/language technology caught up", such that you could make lists/graphs/links. (Actually, I suspect that the access type was /always/ meant to be key: that the Cursor-type in the Ada.Containers.* packages /should/ be access types, discriminated by the container.)

> External references are another problem. You might have not only internal links that can be stored as offsets to the first allocated element, but also a closure of objects your persistent object refers to. Since objects can be shared you need to handle multiple references and maintain a catalogue of objects in the persistent storage. Then you need some policy of updating shared objects.

Honestly, at /that/ point what would be useful is a common Ada+PL/SQL runtime, with the runtime containing a full database-engine.

From: bjorn-lundin
Date: Sat, 5 Jul 2025 10:02:12 +0000

I did. I saw the gem many years ago, and used it to keep track of a sequence number in a communication process. The protocol wants the telegrams ordered by sequence num, which increases by 1 for each tgm. To make the other side happy I saved the sequence num on disk through this pragma, so I did not need to serialize it myself. It is read upon restart which was the point.

That protocol was later changed to accept 0 as a restart marker.

I also customized the storage format. At first it was just 4 integers (32 bit num) but I made it human readable.

It is still running on that site.

Scalar Object 'Valid Attribute

From: klnelson
Subject: Scalar Object 'valid Attribute
Date: Sat, 5 Jul 2025 03:33:45 +0000
Newsgroups: forum.ada-lang.io

I would like to better understand the rationale for the 'Valid attribute for scalar objects as it seems backward to me. Take the case of performing an Unchecked_Conversion to assign an integer value to an enumeration object whose type has a representation clause assigning each member an integer value. The unchecked conversion is performed, assigning the object a (potentially erroneous) value then 'Valid is applied to check if the object is erroneous and if so, the error is handled/corrected.

It seems it would be easier if 'Valid could be applied to the enumeration type and the integer could be checked by doing My_Enum'Valid (My_Int) before doing the assignment and erroneous objects could be avoided all together. This could be wrapped in a simple function:

```
function To_Protocol_ID
  (id : C.unsigned_short)
  return Protocol_ID is
    (if Protocol_ID'Valid (id)
       then Protocol_ID'enum_val (id)
       else Protocol_None);
```

The best general solution I came up with is a Checked_Conversion generic, wrapping Unchecked_Conversion where an enumeration default value is returned in the case where 'Valid returns false. The conversion default is specified at instantiation or as a parameter replacing the instantiated default at checked conversion time.

```
generic
    type Source is ();
    type Target is ();
    Target_Default : Target;
function Checked_Conversion (S : Source;
  Default : Target := Target_Default)
return Target;

function Checked_Conversion (S : Source;
   Default : Target := Target_Default)
return Target is
    function To_Target is new
    Ada.Unchecked_Conversion
    (Source, Target);
    Obj : constant Target := To_Target (S);
begin
    return (if Obj'Valid then Obj
       else Default);
end Checked_Conversion;
```

Is there a better way? If not, why doesn't Ada have Ada.Checked_Conversion?

[Code example omitted. –arm]

From: OneWingedShark
Date: Sat, 5 Jul 2025 04:13:07 +0000

> It seems it would be easier if 'Valid could be applied to the enumeration type and the integer could be checked by doing My_Enum'Valid (My_Int) before doing the assignment and erroneous objects could be avoided all together.

I tend to agree with you, but there are some reasons it is how it is.

IIUC, the reason that it is on the value, rather than the type/subtype, is for things like (a) memory-overlay, and (b) memory-mapped I/O. — I believe the rationale was that you could define your [sub]type with all the correct values and use overlay, mapped-I/O, or unchecked conversion on (e.g.) incoming data, then use Val'Valid on that object, rather than something like:

```
declare
  Raw_Data : Integer with Import,
    Address => IO_Address;
begin
  if Target_Type'Valid( Raw_Data ) then
  -- …
end;
```

As you can see, the problem with a Type'Valid attribute is that it would be hiding a conversion, then testing against the validation, /*then*/ returning a Boolean… meaning that you just lost the passing value if it *is* valid, forcing a reconversion. — we could have had it work on the type, if we'd had attributes that reference functions with an out-parameter, but 'Valid was prior Ada2012.

@sttaft would have more details on it.

From: klnelson
Date: Sat, 5 Jul 2025 16:36:28 +0000

> As you can see, the problem with a Type'Valid attribute is that it would be hiding a conversion, then testing against the validation, /*then*/ returning a Boolean… meaning that you just lost the passing value if it *is* valid, forcing a reconversion. — we could have had it work on the type, if we'd had attributes that reference functions with an out-parameter, but 'Valid was prior Ada2012.

I am not familiar with these type conversion mechanics, though this check was pretty simple. In the case of Protocol_ID, just take the candidate value and look it up in a table containing the valid values of 10, 20, and 30 and return a boolean, avoiding creating an erroneous object and/or raising Constraint_Error. A valid object of type Protocol_ID can then be created without an exception occurring.

IMO, enumeration types have been broken since the language inception as you can't (easily) model values of this nature and it looked like 'Valid, 'Enum_Rep, and 'Enum_Val and resolved this. If someone would be kind enough to point me to the correct section of ARM / rationale document(s), I can just read about it. I've been away from Ada since 2005 and have lost track of the specification documents. A quick google did not provide any results and will try again.

How to 'pass a Function' to a Procedure in Ada?

From: Ret_Build_Engineer
Subject: Very Dumb Review Question: How to 'pass a Function' to a Procedure in Ada?
Date: Mon, 7 Jul 2025 23:38:48 +0000
Newsgroups: forum.ada-lang.io

Please pardon the elementary nature of the question.

I'm trying to remember (I know I did something like this previously), but I can't remember when/where; also I really don't feel that 'pass a function in Ada' is a sufficiently well phrased search question. Meanwhile I guess I'll use it as I can't think of anything better yet.

The end result would be a library of Ada code where someone wants to explore specific results (numerical integration in this case) from an arbitrary function, where the code being exercised has already been provided as a library.

So, how are functions passed as parameters in Ada? Or how the heck does a library know what function it is going to be executing in the future?

Right now, I'm just going to hard-code a function inside the integration procedure.

From: OneWingedShark
Date: Tue, 8 Jul 2025 02:42:43 +0000

> So, how are functions passed as parameters in Ada?

As generic formal parameters.

/Technically/ you can use an access to a subprogram, but generics ought to be preferred over mere accesses.

> Or how the heck does I library know what function it is going to be executing in the future?

For a generic, let me present the problem where you want to insert a print/logging of a value and return that value:

```
generic
   type Value is limited private;
   with Procedure Print( Object : Value)
     is null;
function Debug_Value( X: Value )
   return Value;
function Debug_Value( X: Value )
   return Value is
begin
  return Result : constant Value := X do
    Print( X );
  end return;
end Debug_Value;
```

Here, we are providing a Print function (as well as the type) as a formal parameter, this allows us to instantiate Debug_Value with something like, say, a function of Write_Line(Object'Image).

Note: It may be prudent to have a polynomial type, perhaps implemented as Array (Natural range) of Real, where the elements thereof are the [coefficients of the] exponents.

From: OneWingedShark
Date: Tue, 8 Jul 2025 03:38:06 +0000

Sorry, I meant *"coefficients of the exponents"*.

EG: (3 => 2, 2 => -4, 1 => 7, 0 => 11) would be the array for $2x^3-4x^2+7x^1+11x^0$.

From: Ret_Build_Engineer
Date: Tue, 8 Jul 2025 04:06:54 +0000

Thanks for the clarification regarding the coefficients

Let's say the function in question was in terms of more than one variable…or was a trig function?

I was thinking about defining the function as something like (typing on the fly here):

```
function f (x : float) return float is
begin
  return sin (x); -- potentially much more
                  -- complicated
end f;
```

then the function which is in the library (not yet) which does the math (a numerical integration method in this case) would be given the parameter of 'f', the function.

Again, I really do not feel that I am expressing myself fluently here

From: OneWingedShark
Date: Tue, 8 Jul 2025 06:06:33 +0000

You're getting into potentially tricky territory.

While it is easy to see that integrating sin(x) gives you -cos(x)[+C], the thing that you have to remember is that the computer /isn't/ doing symbol manipulation (well, it /can/, but then you're in for doing the whole of the engine itself, in which case you're probably better off using LISP) — there are ways to tackle the problems, but full-blown numerics is…difficult.

Consider:

```
generic
  with function F( X : Real ) return Real;
package Numeric_Stuff is
  -- Note that by instantiating this, you get a
  -- function which returns a single value, and
  -- is without parameters.
```

```
   generic
      Start, Stop : Real;
   function Generic_Evaluate return Real;
   --…
   Function Generic_Evaluate return Real
     is (F(Stop) - F(Start));
   --…
```

As you can see, there are major consequences of how to use a library based on the choices in design. The above is /probably/ less useful than some other approach, but might be well suited for the particular application.

From: dmitry-kazakov
Date: Tue, 8 Jul 2025 09:11:27 +0000

This is the case when you do not want generics. Normally generics are always the last design choice because of their static nature. You cannot do anything that requires late bindings with them. As it was already pointed out you can use access to subprogram and thus pass any subprogram from anywhere, e.g. from a library loaded /*after*/ your program was compiled.

A more complicated but also more flexible way is to use methods. This approach is actively deployed in various software patterns, e.g. in the visitor pattern when you need to call something when iterating something while maintaining some state external to the iterator itself. In its simple form you pass an object of a type that has a primitive operation which is then called back. The primitive operation is your function. Now this is also static as you need to override the operation in order to put your function body in there. But you can make it dynamic using an access discriminant. So you pass your function as a discriminant of:

```
type Functor
   (F : access function (X : Float)
      return Float
   ) is tagged null record;
function Call_Me (Object : Functor;
   X : Float) return Float;
```

Now let your integrator declared as:

```
function Integrate (What : Functor;
   From, To : Float);
```

Then you can use it as:

```
   Integrate (Functor (Sin'Access), 0.1, 0.5);
```

From: jere
Date: Tue, 8 Jul 2025 20:14:07 +0000

Everyone has already discussed pros and cons. Here are a few examples in compilable code if you need.

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Integer_Text_IO; use
   Ada.Integer_Text_IO;
procedure Test is
   -- WAY #1
   -- Named subprogram access type
   -- parameter is used when you want to
   -- "save" the passed in function for later
   type Function_Access is access
      function(Item : Integer) return Integer;
   function f1(f : Function_Access)
      return Integer;

   -- WAY #2
   -- Anonymous access parameter is used
   -- when you just want to execute
   -- it as soon as possible
   function f2(f : not null access
      function(Item : Integer) return Integer)
      return Integer;

   -- WAY #3
   -- Generic is another option
   generic
      with function f(Item : Integer)
         return Integer;
   function f3_generic return Integer;

   -- Junk implementations just for testing
   function f1(f : Function_Access)
      return Integer is (f(100));
   function f2(f : not null access
      function(Item : Integer) return Integer)
      return Integer is (f(100));
   function f3_generic
      return Integer is (f(100));

   -- Here's the function to pass in
   function Test(Item : Integer)
      return Integer is (2*Item);

   -- Junk variables to hold results
   v1 : Integer := f1(Test'Access);
   v2 : Integer := f2(Test'Access);

   function f3 is new f3_generic(Test);

   v3 : Integer := f3;

begin
   null;
end Test;
```

Finalize/Adjust Raised Exception

From: MarioBlunk
Subject: Raised Program_error : Finalize/Adjust Raised Exception
Date: Thu, 10 Jul 2025 08:28:41 +0000
Newsgroups: forum.ada-lang.io

After updating to GNAT 15.1.1 20250626 I get an exception in connection with controlled types. I have a function that returns a controlled type. Before the return statement I get the above mentioned exception.

So before providing more details, my question is whether anyone here is experiencing the same trouble?

From: dmitry-kazakov
Date: Thu, 10 Jul 2025 08:53:37 +0000

This one of the most common bugs Ada programmer faces. Controlled types are all OK, you need to check your implementations of Finalize and Adjust for unhandled exceptions.

See also GNAT.Exception_Actions for tracing exceptions. Program_Error just an indication of some chain of exceptions prior to it.

From: jere
Date: Thu, 10 Jul 2025 13:13:39 +0000

Also keep in mind that GNAT itself has some bugs with that specific scenario. I've had standard Ada containers raise that exception (with no overrides from me). Recently I had to wrap an indefinite holder instance in a separate controlled type and catch the exception in a custom finalize operation to avoid it.

I haven't tried v15 yet though, this is all pre 15 experience on my part.

GUI Libraries - Suggestions

From: vjalmr
Subject: Gui Libraries - Suggestions
Date: Sat, 12 Jul 2025 17:36:38 +0000
Newsgroups: forum.ada-lang.io

My current scenario: I need to write an application with a GUI that works on *nix, win, and osx (or whatever apple calls it these days).

What is my best option for this, in regards to GUI? Could I achieve this by using GTK? I have never seen GTK running on win or osx, so any and all comments would be appreciated.

Please, /no/ web tech.

From: dmitry-kazakov
Date: Sat, 12 Jul 2025 19:31:48 +0000

In case you didn't know GNAT Studio (GPS) is GTK3. So yes, Windows and BSD (Mac stuff) are supported.

The choice is usually Qt or GTK. GTK tries to use look-and-feel of the host OS. Qt is available on a wider range of OSes. Another difference is that Qt is C++ and GTK is C. C++ is far more difficult to communicate.

In any case both have Ada bindings.

From: jere
Date: Sat, 12 Jul 2025 21:28:35 +0000

I use Gnoga [1] a lot. It's pretty handy at work for making some quick simulators and things when I want a GUI. It's browser based, so that helps with cross platform scenarios. I only use Windows and Linux, but so far all of my stuff has compiled for both pretty easily.

[1] https://github.com/Blady-Com/gnoga

From: Fabien.C
Date: Tue, 15 Jul 2025 08:39:14 +0000

I am eagerly waiting for the release of LibreFrame: https://www.libreframe.com/

I would love to see if we can make a "native" rendering front-end for it.

Starting with something like RayGUI and maybe move to a full Ada implementation afterwards.

(I don't know if the Sowebio people are hanging out around here)

From: dmitry-kazakov
Date: Tue, 15 Jul 2025 09:19:20 +0000

Interesting, but does not look like a GUI at all.

To me GUI is all about widgets and handling events/messages. The problem with existing GUI like Qt, GTK, Win32 etc is abstraction inversion since handling events happens in wrong places. E.g. button press is handled by button widget rather than by a container of.

The event processing is unstructured and requires to be in front while the problem space logic is pushed back into callbacks.

Then of course, there is a push vs. pull problem. GUIs are built event-driven. This architecture is totally unsuitable for real-time, high-frequency and massive amounts of data applications where you would rather use polling. You can of course create an oscilloscope widget. I did it in GTK and Win32, but it requires a lot of acrobatics.

From: NumberSix
Date: Tue, 15 Jul 2025 14:57:54 +0000

We're trying a different approach. Give us a chance to show you something consistent. Time doesn't respect what's done without it. And there are, for the moment, only two of us.

For now, we're taking care of the foundations by rewriting the websocket interface and using epoll. We're now up to 1.5 M req/s on average hardware. But we can do even better. And we'll do it if it makes sense. Like automatic reconnection, since websocket is obviously sensitive in this respect.

As this is still being settled, we also have a schedule to complete. Building - LibreFrame [1]

Thanks for waiting a few months, but we'll already have things ready between September and October this year.

We are, of course, always open to comments. We're really counting on it.

All the best from Oleron Island.
Stéphane

[1] https://www.libreframe.com/building

From: dmitry-kazakov
Date: Tue, 15 Jul 2025 15:36:04 +0000

Ada GUI project might be very interesting. Though it is a mammoth task I am not even sure how to approach it avoiding typical errors of GUI architectures.

Mentioning WebSockets in that context rings alarm bells to me. Anyway, good luck.

From: NumberSix
Date: Thu, 17 Jul 2025 15:58:16 +0000

Yes, websocket is natively very different from what we're used to on the web.

However, the use of websockets is a key feature of Web GUI LibreFrame architecture.

In the age of fast, universal networks, permanent bidirectional communication (with the necessary precautions) could greatly enhance the user experience, compared with the classic transactional HTTP, with tons of JS…

To discuss this in more detail, I'd have to talk about LibreFrame's complete architecture, which can be summed up as "everything from the server". Indeed, in our concept, the amount of client-side JS is negligible.

To be user friendly, LibreFrame retains the entire user session, even across multiple tabs. LibreFrame emulates HTTP navigation. Arrows to return to a previous page or move forward are supported.

Similarly, logging out from a session with 10 tabs automatically closes the 10 views. It's a security enhancement. Logging again automatically restores them to the page they were on. We've already demonstrated this at AEiC 2025.

As the icing on the cake, LibreFrame's websockets communication protocol is designed to be very lightweight, and even works on 64 Kbps links (i.e. 0.064 Mbps).

So, may be WebSockets, properly managed, could ring you some pleasant melody

And, yes, you're absolutely right. LibreFrame, as a whole, is a mammoth task. But we're tough. This project originated in 2020 with v20 library. By 2023-2024, we've reached a lot of milestones with v22 framework, checked our assumptions, drawn up a schedule, and we're moving on.

LibreFrame 1.0 will be on show at FOSDEM 2026, with applications already built using it.

We're really looking forward to this first version to gather feedback.

From: elimliadam
Date: Thu, 17 Jul 2025 17:52:40 +0000

FLTKAda [1]

[1] http://www.jedbarber.id.au/cgi-bin/cgit.cgi/fltkada/about/

From: docandrew
Date: Fri, 18 Jul 2025 21:40:18 +0000

I did a little experiment with a native Ada IMGUI concept here:

GitHub - docandrew/DAGBuild: Modern Build Tool

https://github.com/docandrew/dagbuild

It ended up just being a science project but might be interesting to folks here.

Uses the Ada SDL backend.

Hosting an Ada Web App

From: patrickjh
Subject: Hosting an Ada Web App
Date: Sun, 13 Jul 2025 07:05:54 +0000
Newsgroups: forum.ada-lang.io

I am exploring the idea of using Ada to write a web backend mainly because it seems like a nice language. I am just curious - do you have any thoughts about the best way to write and host Ada code for the cloud? I can think of some possibilities and am looking for feedback on which of these might be a good idea and which are a bad idea. Any other ideas are welcome too:

Idea 1: Write the Ada code as CGI scripts running on a Linux cloud virtual machine using something like Lighttpd. Seems simple enough, but I would have to sys admin a Linux system and CGI is kinda low performance.

Idea 2: Use an Ada web framework and run the Ada code as a process on a Linux cloud virtual machine. Also seems simple enough but I would have to sys admin a Linux system.

Idea 3: Write some serverless functions for AWS Lambda in Ada. Similar to the CGI idea the code would be simple but I would not have to sys admin anything. Has anyone done this before? Seems a bit tricky as Ada is not one of the officially supported languages and apparently you have to create some kind of container image.

Idea 4: Since Ada can be used for embedded use cases, is there maybe a way to create a VM image of Ada code that can run as a web server on something like EC2 without any operating system? It would probably have challenges but I imagine that if Ada can run on baremetal hardware without an OS, there might be a way to run in the cloud without an OS?

Idea 5: Create a Docker Container Image with the Ada binary and use that with some cloud service like Kubernetes. There do seem to be some Docker containers for Ada, like this container, → https://hub.docker.com/r/esolang/ada. Has anyone used anything like that?

From: NumberSix
Date: Mon, 14 Jul 2025 09:38:53 +0000

You can give a try to:

Transactional (http)

AWA: Ada Web Application [1] from Stéphane Carrez. It's based on Ada Web Server, comes with an ORM and a lot of goodies.

Bi-directional (ws)

Now (which is the past somewhere), our POC called v22, with real industrial apps created with it, running 365/214: https:/v22.soweb.io

It's based on Gnoga and comes with a highly effective overlay to boost productivity and everything you need to make a complete application.

The future (coming quickly): https://www.libreframe.com

Based on v22 but replacing Gnoga with something completely new. After a presentation at AEIC2025 in Paris, which used an existing Ada websocket layer, we are now recoding this part, for greater simplicity and performance. Our initial feedback has been excellent (speed gains of over x200 and 1,5M req/s on an average server!), and we are now recoding the http web server (websocket starts from http 1.1). When that's finished, we'll extend the available widgets and theming. Expect all this by the end of the year.

As far as CGI is concerned, Matreska has a Fast CGI implementation which, when coupled with a Fast_CGI cache via Nginx, delivers extraordinary performance, but this only applies to static pages.

On the production side, we're not seduced by docker (old technology with a new look) and kubernetes (too complex and irrelevant in our case). We don't use microservices (inefficient and difficult to debug) and we don't use the cloud (too expensive and not so reliable).

We host our own infrastructure on real bare-metal servers, using the Xen hypervisor in PVH mode (no qemu, full para-virtualization and bare metal speed), with Debian everywhere…

There are other ways of doing things, it's just ours. If you feel more comfortable with Docker, that's not a problem.

Administering a Linux server is no more difficult than administering the cloud. On the other hand, a Linux server is much cheaper to run, especially if you've virtualized it so you can have lots of instances. A little effort up front for a big financial gain and greater autonomy.

[1] https://ada-awa.readthedocs.io/en/latest/

From: stcarrez
Date: Mon, 14 Jul 2025 18:52:36 +0000

Some information for you:

- The Ada France [1] web site is running on a (tiny) bare-metal Linux and the web server is based on AWA+AWS. The database used is MySQL. Sources are available at [2]
- My web site https://blog.vacs.fr is also running on bare-metal Linux with AWA+AWS. The Ada binary behind it contains 3 AWA applications that are separate to serve different web sites (virtual hosts). The database used is MySQL.
- The Porion build manager [3] is another AWA+AWS server but it runs within a virtual machine. The database used is SQLite (it could be PostgreSQL or MySQL). Sources are available at [4]
- HTML/Wiki to Wiki Converter [5] is another AWS+Ada Servlet server (a small subset of AWA). It runs within a Docker container. Sources are available at: GitHub - stcarrez/wi2wic: Wiki 2 Wiki Converter [6]
- To show some features of AWA, I created Atlas [7] and you can run it from a Docker container. You'll find some information there on how to start/stop the container.

Docker and Kubernetes bring another level of complexity and you can probably leave that later in the deployment phase.

[1] https://www.ada-france.org/

[2] https://github.com/Ada-France/ada-france

[3] https://porion.vacs.fr

[4] https://gitlab.com/stcarrez/porion

[5] https://wi2wic.vacs.fr/wi2wic/index.html

[6] https://github.com/stcarrez/wi2wic

[7] https://github.com/stcarrez/atlas

Experiences with Large Projects

From: cowile
Subject: Experiences with Large Projects
Date: Tue, 22 Jul 2025 04:20:13 +0000
Newsgroups: forum.ada-lang.io

Ada is often represented as a language for programming in the large. Can someone here with experience in large Ada projects talk about what the experience was working on large Ada projects. What was different from large projects in other languages? What were the primary pains?

Large may be defined generously however you choose, but I'm imagining projects with one or more of:

- Dozens of contributors
- Hundreds of thousands to millions of lines of code
- Production critical for years
- Expansive and changing scope
- Stringent performance requirements such as latency or throughput goals

From: Irvise
Date: Tue, 22 Jul 2025 17:48:38 +0000

I cannot answer this question from first hand experience, however, I do know quite a few Ada programmers working in very large codebases (+2 million lines).

TL;DR: it is just great!

About your points:

- Dozens of contributors: I guess that having a good SCM would help manage all the people. Nevertheless, I assume you are referring to the typical case of "Bob changed X things without notifying Alice, will things break?".
- Ada's very strong typing system, the Compilers sanity checks, extra features (such as liquid types and contracts [1]) and the obvious use of SPARK; will ensure that either random wrong changes do not compile, get caught very early (runtime checks during testing or production) or simply are not allowed by SPARK (the prover).
- A lot of SLOCs: Ada was not a pioneer of the module system, but back in the 80s it was one of the first and it was quite useful and natural. Segmenting large projects by modules/functionality is very easy, recommended and scalable. Newer PLs (read, after 2010s) have realised how important this is and they are also using modules (see Rust). Even C++ finally got some basic module functionality! Just 40 years after Ada… Maybe using C as the baseline design was not a good idea… For Rust, maybe creating an improved version of C++ was also not a good idea…
- Ada was created specifically to deal with very large projects. Eurocontrol [2] deploys a 2.2 million lines program written in Ada (here you can see some insights on its extreme performance tracking [3]). NVIDIA just recently announced that they got ASIL-D certification for their 7 million OS written in Ada/SPARK [4]
- Production critical: pleaaaaase, is this even a question?? See the above answer and also take a look at the Ariane rocket program, Airbus, Boeing, Paris Metro, etc.
- The modular system of Ada allows changes to be contained. Breaking changes (read requirements that change scope) can be detected early by the compiler if they are trivial. The techniques mentioned in the bullet point of point 1. should provide even more assurance and early detection of clashes between changes. If you use generics, type reflection/attributes, etc; a simple change in a source file will automatically expand throughout the rest of your program without you having to do anything!
- Ada generates incredibly performant code. full-stop. Really! Take string (base type, so stl:: bullshit), they are not null/\0 terminated, their length is part of the type itself. The algorithms on structures whose size is always know are incredibly performant. Use Godbolt/Comiler Explorer [5] to see how Ada code compares to other languages. Also, take a look at the bullet point for the second point of yours, that should give you a really good idea on how deep you can go with Ada.
- Just a word of caution, use user defined types for your data, this may help the compiler generate substantially more performant code. If you see that the performance/assembly of Ada is substantially worse than that of C/C++/Rust, it may be that these

languages allow for undefined behaviour and they are taking advantage of it while Ada may be forbidding it!

I hope this answers your points. Please, post any questions, issues or anecdotes that you may have while learning Ada. A great place to get started learning it is https://learn.adacore.com/

[1] https://fosdem.org/2025/schedule/event/fosdem-2025-4879-understanding-liquid-types-contracts-and-formal-verification-with-ada-spark/

[2] https://www.eurocontrol.int/

[3] https://archive.fosdem.org/2020/schedule/event/ada_performance/

[4] https://www.eenewseurope.com/en/nvidia-drives-ada-and-spark-into-driverless-cars/

[5] https://godbolt.org/

From: cowile
Date: Thu, 24 Jul 2025 04:26:57 +0000

Thanks for the answers!

> Ada was created specifically to deal with very large projects. Eurocontrol [1] deploys a 2.2 million lines program written in Ada (here you can see some insights on its extreme performance tracking [2]). NVIDIA just recently announced that they got ASIL-D certification for their 7 million OS written in Ada/SPARK [3]

Would you happen to know anything like how fast those projects build or how much unit testing there is compared to relying on the compiler and formal methods?

[...]

> Another general difference is that you need to be much less defensive (it is due to Ada's type system but also myriads of details like the syntax of the *for* loop, where you can't do dumb things like writing the exit condition wrong, having jump steps of 2 or tampering the loop parameter)

This was something I definitely appreciated about Ada. Some categories of mistake are eliminated by thoughtful syntax, like you have to explicitly specify what block you are closing with end if; or end loop;. That's a big improvement over C's "brackets optional for one line rule". And the additional characters don't bother me because it's more comfortable to type than bracket langs.

> The downside of this is that sometimes you'll see code generators to get around a lot of boilerplate (like in AUnit for unit testing).

> General nestability of packages/functions can lead to insanity

Thanks. These were the kind of things I was looking for when asking about pain points, though the latter seems like a problem in any language.

From: Irvise
Date: Thu, 24 Jul 2025 09:34:27 +0000

> Would you happen to know anything like how fast those projects build or how much unit testing there is compared to relying on the compiler and formal methods?

Ada is quite modular, so once you build the entire project, the rest should just be incremental builds. There should be little to no cascade effects.

NVIDIA used the formal methods for their OS extensively. Eurocontrol, AFAIK, uses ASIS [1]. Sadly, ASIS does not support newer Ada versions and was removed from GCC in v10 (though they still get support from AdaCore as they pay for their tools). ASIS was an extended set of checks done by the compiler and behaved like an extended target/language. AdaCore would like people to start using LibAdalang [2] instead. Also, keep in mind that Ada, when compiled with GCC or LLVM (oh, yes, we have an LLVM backend! [3]) it can use the sanity checks and tooling that they offer, on top of the language spec. You can also use any other analysis tools, such as Valgrind, etc.

Additionally, when dealing with memory, you may not be aware that Ada supports "Memory Pools" (known as arenas in other langs) since 2012. They can also serve as an advanced tool to deal with safety, increase performance and debugability. You can learn much more about the Ada memory model in this outstanding presentation [4] by Jean-Pierre. Also, keep in mind that GNAT offers finalizable types (not yet standard) since V15, which are basically simple controlled types that are compatible with embedded systems, have a much better defined behaviour and can be analyzed by SPARK. Also, SPARK can now prove complete memory correctness ala borrow checker from Rust (including leaks and lifetimes, unlike Rust).

> But since I don't see a lot of opportunities to work on big projects with other people, I created this thread to ask about people that had. I contribute to the Linux kernel as part of my day job and am very interested in OS dev [...]

There are quite a few interesting OS projects in Ada. The biggest and probably the one you would like to dive deep into is Ironclad [5] with its Gloire distribution [6]. Ironclad is a *NIX kernel written in Ada/SPARK, similar to Linux and it can run now a ton of typical *NIX programs, such as videogames . It is not huge, it is only about 40k lines, but it can already do a lot. It also implements its own RTS (RunTime System). You can see an introduction to it here [7]. Oh, and this is mostly written by a couple of 25(±) year olds!

Another interesting OS is HiRTOS [8], it is a hard RT kernel written in SPARK and fully formally verifiable for ARM/RISC-V devices. This one is about the same size as Ironclad, but completely different in nature, targets and design.

Finally, a commercial open source Ada/SPARK micro/separation kernel is Muen [9], we know from disclosed information that it is being used in encryption devices and telecommunications/telephone systems. After a clone of the kernel with all of its submodules, there are 158 kLOCs of Ada (the kernel is only 6200 LOCs).

> I remember trying something like that about ten years ago during a previous spark (lol) of Ada interest, but ran into the problem of not knowing how to cut down the language runtime to work on bare metal and gave up for lack of finding good resources for how to do it.

Wait, what? The runtimes can be zero/light (no runtime whatsoever), extremely basic (allowing for secondary stack management, memory features…), light-tasking (think of the restricted tasking profiles Ravenscar and Jorvik [10], see section D.13) and full-tasking (Windblows, Macs, Linux *BSDs…). You do not have to cut the Runtime, you only have to use the features that can be used with the runtime you are targeting.

For example, if you want to use a light runtime, you must simply not use any tasking or advanced memory thingies. You can take a look at the impressive, great, wonderful and amazing SweetAda [11] project. In total it has 100 kLOC lines. It is a framework to create bare-metal applications in a f*ck ton of boards and architectures. It truly is an amazing project and it is done by a single incredible person. Runtime-wise, it only supports zero/light and secondary stack (selected arches) runtimes, though the goal is to support tasking. It also has some drivers for common peripherals and general virtual thingies, such as FAT support.

Another great project is bb-runtimes [12], this is a collection of arches/boards/fpgas with several runtimes. There are several that have light-tasking or even full-tasking runtimes. It also brings in some peripherals to the mix. This is probably the most complete Runtime-filled repo out there.

You can also check the runtimes for Ironclad, HiRTOS and Muen if you are interested. Also, a nice bit of trivia/updates, the Ravenscar and Jorvik profiles can be verified by SPARK, meaning that if you stick to them, SPARK can prove the correct behaviour of your multitasking applications [13]!

[1] https://www.adacore.com/asis

[2] https://github.com/AdaCore/libadalang

[3] https://github.com/AdaCore/gnat-llvm

[4] https://archive.fosdem.org/2016/schedule/event/ada_memory/

[5] https://ironclad-os.org/

[6] https://codeberg.org/Ironclad/Gloire

[7] https://fosdem.org/2025/schedule/event/fosdem-2025-4871-developing-device-drivers-for-ironclad-using-ada/

[8] https://github.com/jgrivera67/HiRTOS

[9] https://muen.codelabs.ch/

[10] https://www.adaic.org/resources/add_content/standards/22rm/rm-final.pdf

[11] https://github.com/gabriele-galeotti/SweetAda

[12] https://github.com/AdaCore/bb-runtimes

[13] https://docs.adacore.com/spark2014-docs/html/ug/en/source/concurrency.html

From: cowile
Date: Fri, 25 Jul 2025 06:08:52 +0000

[...]

> There's a killer feature I never see anyone talk about: You can specify dependencies between Ada packages for initialization during the environment task prior to the entry procedure, to avoid the static initialization order fiasco [1].

Interesting. Where can I read more about this feature?

> I'm wondering if the era of programming languages focusing on "expressiveness" is over, and those focusing on semantic correctness (like Ada) is now here.

Even before LLMs I noticed a (re)surging interest in strong, static type systems. Haskell gained some popularity in the mid-2010s before falling off (presumably happy to stay an ivory-tower language avoiding success at all costs) and Rust keeps growing. I think everyone is running from the horrors of the scripting language dominated web era. My view from inside Big Tech is the dominant dynamic languages JavaScript and Python are banned in their naked form and must use typed variants.

Ada has benefited from this trend and IMO ends up looking like a language ahead of its time. And the over-complexity criticisms don't stick anymore because if anything Ada feels simpler than a lot of more modern languages.

There will absolutely be synergy between LLMs and languages with strong correctness focus. Like you say it is easier to section off the vibe coded bits and good compiler feedback helps keep the LLMs on track for longer tasks. With LLms, the thing most pushing people towards "expressive" languages, the extra typing, is a non issue.

[1] https://en.cppreference.com/w/cpp/language/siof.html

From: Irvise
Date: Fri, 25 Jul 2025 10:19:27 +0000

> I'm also curious about any support that makes debugging easier. Does it play well with GDB and are there other more useful debugging tools out there?

GDB has built in support for Ada OOTB and it works wonders. I always launch GDB with catch assertions and catch exceptions and just let my Ada program (built with checks enabled, which is the default in GCC) and let it tell me where things fail. It literally is the most pleasurable experience I have ever had with debugging. Additionally, it supports all the features of GDB, so you can also do remote debugging with OpenOCD in embedded boards. I have used it with great success. SweetAda has built in support for it for its targets. Also, obviously, you can debug QEMU binaries, etc. Ada is no different than C/C++ and I would dare to say it has even better features than them because of the language design and because of the extra small features in GDB. Also, you can deploy an Ada program in Renode [1] if you wanna give them a test without the real hardware. Here is a small blog detailing Ada and Renode [2].

> Interesting. Where can I read more about this feature?

This is a good place [3]

> Ada has benefited from this trend and IMO ends up looking like a language ahead of its time. And the over-complexity criticisms don't stick anymore because if anything Ada feels simpler than a lot of more modern languages.

110% right!

[1] https://renode.io/

[2] https://blog.adacore.com/getting-started-with-renode-simulating-an-ada-stm32f429disco-blinky-firmware

[3] https://gcc.gnu.org/onlinedocs/gnat_ugn/Controlling-the-Elaboration-Order-in-Ada.html

From: kevlar700
Date: Fri, 25 Jul 2025 10:33:19 +0000

> There is no documentation for the "zero"/light runtimes because… that is what they represent, the absolute lack of any runtime x)

There is some runtime support in the light runtimes and even zero. It is just all independent of clocks etc. and so can be made to run on any chip much more easily and on chip families. For example array bounds, 'Image, 'Value etc.

5. Predefined GNAT Pro Run-Times [1]

Hopefully these may be of some use for Linux but I find Linux drivers a pain to read with all the abstractions never mind trying to integrate Ada code for many targets so good luck.

Hacking the Linux Kernel in Ada - Part 1 - Linux.com [2]

For this three part series, we implemented a 'pedal to the metal' GPIO driven, flashing of a LED, in the context of a Linux kernel module for the NVIDIA Jetson Nano development board (kernel-based v4.9.294, arm64) in my favorite programming language...

Ada on any ARM Cortex-M device, in just a couple minutes [3]

GitHub - alkhimey/Ada_Kernel_Module_Framework: Framework for writing Linux kernel modules in Ada [4]

GitHub - AdaCore/bb-runtimes: Source repository for the GNAT Bare Metal BSPs [5]

[1] https://docs.adacore.com/gnat_ugx-docs/html/gnat_ugx/gnat_ugx/gnat_runtimes.html

[2] https://www.linux.com/audience/developers/hacking-the-linux-kernel-in-ada-part-1/

[3] https://blog.adacore.com/ada-on-any-arm-cortex-m-device-in-just-a-couple-minutes

[4] https://github.com/alkhimey/Ada_Kernel_Module_Framework

[5] https://github.com/AdaCore/bb-runtimes

From: pyj
Date: Thu, 24 Jul 2025 11:59:47 +0000

[...]

> how fast those projects build or how much unit testing there is compared to relying on the compiler and formal methods?

IME, just the baseline Ada experience prevents you from shooting yourself in the foot 95% of the time. e.g. you cannot return a reference to an in and out param, or a stack variable unless you explicitly mark them as aliased.

Build times are pretty good compared to "I changed a low level header file and had to wait 30 minutes for a build." I have heard extensive formal methods proofs can take a long time (hours), though GNAT studio has ways to target analyses. My verified postfix calculator [2] takes about a minute or so on a 5950x, but proofs seem to parallelize really well. Trendy Test runs all tests randomized and in parallel, the problem being if you have unit tests depending on module state, the

result isn't well defined unless you mark those tests as "run in serial."

> contributing an Alire package called format_strings (AI-assisted)

I've been playing around more with LLMs and vibe coded decent Ada versions of Rust's Rc and Box equivalents in about 10 minutes. A big reason I still do Ada hobby dev is because the spec/implementation firewall gives a place to cordon off vibe-coded elements until you have time to go through them. I'm wondering if the era of programming languages focusing on "expressiveness" is over, and those focusing on semantic correctness (like Ada) is now here.

[1] https://en.cppreference.com/w/cpp/language/siof.html

[2] https://pyjarrett.github.io/2025/06/10/postfix-calculator.html

From: zertovitch
Date: Sun, 27 Jul 2025 16:20:38 +0000

> Would you happen to know anything like how fast those projects build

On a "normal" PC a ~3 million LOC project takes a few seconds to build when you are back after, say, an absence of one week and catch up with other people's changes. For a full build (usually unnecessary), it depends on the PC: a 4-core takes 7 minutes, a 24-core takes 45 seconds.

From: zertovitch
Date: Tue, 22 Jul 2025 19:32:11 +0000

The *big* difference with language X (choose one with conditional compilation, "include"'s, global scope by default, pointers everywhere, soft typing, …) is *Modularity*!

The bigger the teams and project size, the bigger the difference. While you get the whole modularity integrated with Ada and have minimal recompilation at each build, the X guys wait for whole rebuilds each time they change a curly bracket. Plus using, configuring, scripting external tools that try to figure out dependencies - it's a lot of work for them!

Then, Ada's waterproof type system and the low usage of pointers. The X guys worry about getting memory dumps, doing forensic about that, etc.

Another general difference is that you need to be much less defensive (it is due to Ada's type system but also myriads of details like the syntax of the *for* loop, where you can't do dumb things like writing the exit condition wrong, having jump steps of 2 or tampering the loop parameter).

This confidence is both a blessing and (potentially) a curse: with Ada you are much more relaxed thanks to the early bug detection. But you (individually or collectively) may even become /too/ relaxed and a bit spoiled. That's the potential curse.

From: JC001
Date: Wed, 23 Jul 2025 06:32:04 +0000

> But you … may even become /too/ relaxed and a bit spoiled.

> Ada has made you lazy and careless. You can write programs in C that are just as safe by the simple application of super-human diligence.

E. Robert Tisdale

From: JC001
Date: Wed, 23 Jul 2025 06:29:13 +0000

My experience is similar to zertovich's. Starting with the architecture phase, and continuing throughout the design phase, design decisions are formalized in package specifications. These are then enforced by the compiler when the project gets to code generation. In effect, there was no software integration effort needed, since it had been done by the compiler during implementation. Compared to other languages, especially those that would happily compile obsolete calls to subprograms whose specifications had changed, this represented significant savings.

Of course, I've also seen projects that treated package specifications like C header files: an inconvenience required by the compiler, but not something that anyone ever read. They also tended not to use user-defined numeric types, and made heavy use of access types. They did not see as much advantage from Ada.

From: pyj
Date: Wed, 23 Jul 2025 12:07:40 +0000

My normal day job is jumping into large codebases (millions of lines, usually C++) and helping out. [...]

> What was different from large projects in other languages?

1. A lot of problem detail is better hidden in packages due to the language structure.

Ada splits code into specification and body implementations, when written for GNAT each is in a separate file (.ads and .adb respectively). This, combined with encapsulation in Ada being at the package level (types are opaque or non-opaque, visibility is based on package relation), not the class level (C++, C#) means a lot of implementation details stay in the related implementation files. It also means types related to the same subproblem will be in the same file, and you don't need to hop between multiple files to see the entire picture. 99% of the time, just seeing the spec file is all you need, and that spec file is much shorter.

In C++, C#, or Java, you start peeling back classes and interfaces and it takes a while to get to an executable line of code, because you need a constellation of objects to do anything meaningful. In Ada, I find that subproblems end up as distinct packages and the focus is on functional behavior, not on playing connect the dots with types.

In C++ or Rust, you'll often deal with related types (types within types). In Ada, types cannot declare additional types inside themselves, everything is declared at the package level, so I often find an entire subproblem solved in the implementation with all the types used there, rather than jumping five files to understand a high level flow path.

2. A lack of macros and Turing-complete templates means less "magic code."

I'm confident I could take almost any programmer and toss them into an Ada codebase and they'd be able to read almost any part of the entire codebase in a few weeks. There's only a few secret handshakes in generics code [...] that you have to look up.

C++ and Rust code lets you really shoot yourself in the foot with complicated macros. Ruby on Rails is built on this sort of magic with metaprogramming as well. Rails, C++ and Rust do a lot of cool stuff with the macros and metaprogramming, but it can hinder understanding what is actually going on.

Ada doesn't have macros, so what you read is what you get. There's no wild __VA_ARGS__ usage in macros or std::enable_if SFINAE or macro_rules! code with wild and different rules. Ada generics occur at the package and function (or procedure) level, so the little bit of "this is what we're generic over" is the weird part and the rest is "just code." You don't end in angle-bracket hell, instead you get a different type of hell where you have to explicitly instantiate templates, but then you use them just like non-generic code.

The downside of this is that sometimes you'll see code generators to get around a lot of boilerplate (like in AUnit for unit testing).

Another downside is that straightforwardness sometimes just isn't convenient. In Trendy Test, I abuse the exception mechanism for flow control in various ways to handle test registration and running. If I had the macro power of C++ or Ruby, or the metaprogramming of Ruby, there would be much better ergonomics.

3. Constraints help

You can embed a lot of information in Ada directly. Two different ints might have different semantic meaning and shouldn't be mixed and Ada lets you do that, like a built-in Rust newtype or a Go type-definition. Ada depends on function overloading, so you can use that system to only allow meaningful operations: it makes sense to multiply Meters_Per_Second by Seconds to get Meters. If you screw something up with a derived type, the compiler has your back

and it embeds the meaning into the program.

This also goes for pre/post conditions which are on the specification, not an assert hidden inside an implementation. Often I didn't have to bother looking at the implementation of something, which saved time.

4. General nestability of packages/functions can lead to insanity

The one big issue I've seen in Ada is the ability to nest functions (using that term here instead of "subprogram") inside of functions combined with the lack of closures in the language. One project I was looking to contribute to had multiply-nested local functions, with a gigantic function with most contents starting halfway across my editor.

This is exaggerated, but something like this:

```
procedure Foo is
  procedure Foo1 is
    declare
      procedure Foo2 is
        procedure Foo3 is
          -- ...
    begin
      -- ... some actual executable code
```

[1] https://ada-lang.io/docs/arm/

From: zertovitch
Date: Wed, 23 Jul 2025 18:25:17 +0000

> In Ada, types cannot declare additional types inside themselves, everything is declared at the package level

It depends on what you mean by "/the/ package level": every declaration can be local (to packages or subprograms): it includes types, generics, …, and that, within nested packages, functions, etc.

You can have an Ada program without packages but with custom types:

```
with Ada.Text_IO;
procedure Hello is
  type ABC is (a, b, c);

  procedure Nested (p : ABC) is

    type HW is (hello, world);

  begin
    for i in HW loop
      Ada.Text_IO.Put (p'Image & ' ' & i'Image
& ' ');
    end loop;
  end;

begin
  for p in ABC loop
    Nested (p);
  end loop;
end Hello;
```

> General nestability of packages/functions can lead to insanity

But on the bright side, nesting is very practical. Some subprograms make sense only within a certain context. You also save tons of parameters - an advantage in readability /and/ performance.

Of course each feature can be misued.

Note that AdaControl has rules about limiting the nesting:

> Max_Nesting (x4) Controls scopes nested more deeply than a given limit.

From: pyj
Date: Thu, 24 Jul 2025 00:40:38 +0000

> It depends on what you mean by "/the/ package level": every declaration can be local (to packages or subprograms): it includes types, generics, …, and that, within nested packages, functions, etc.

You're exactly right.

What I was trying to do was contrast to nested types like in C++ (struct Foo { struct Bar {}; };). You can declare types inside declare or function or procedure or task bodies, but only those inside packages are visible from outside their containing elements AFAIK, and hence usable when from outside and hence part of the programming boundary interface. They are also visible inside nested subprograms.

C++ and Rust really do seem to love their related types (like $whatever_container::iterator, $whatever_type::reference) whereas in Ada these types would likely be declared at the same level (Ada.Containers.Vectors.Iterator, Ada.Containers.Vectors.Reference_Type)

> Some subprograms make sense only within a certain context. You also save tons of parameters - an advantage in readability /and/ performance. Of course each feature can be misused.

Helper subprograms definitely make sense in some circumstances. I find it hard to read more than one set of lexically nested subprograms. For efficiency, it depends on how many parameters, how often you're calling them, the weight of your parameters, if the compiler will inline and what the compiler actually outputs. In this case I'm not sure, I'd have to find or write something with deep nesting subprograms, measure and look at some assembly.

From: kevlar700
Date: Fri, 25 Jul 2025 10:42:33 +0000

Ada was designed with reducing project lifetime costs of large projects as a primary goal. I'm not sure if any other language was actually even designed as such but certainly not so carefully.

Comparing the development costs and other benefits of Ada or SPARK vs other languages

https://forum.ada-lang.io/t/comparing-the-development-costs-and-other-benefits-of-ada-or-spark-vs-other-languages/681

I can't even find a date on it [1] but it seems some were against Ada quite early on. I guess they liked getting monopolies by using application specific languages. Wow were they grasping at straws if this is the best they could come up with. Some features should only be used sparingly to alleviate a particular pain point and some have been improved upon since then too.

[1] https://dl.acm.org/doi/pdf/10.1145/1041326.1041327

From: pyj
Date: Fri, 25 Jul 2025 23:35:08 +0000

> Ada was designed with reducing project lifetime costs of large projects as a primary goal.

> Perhaps the most astounding example is the YF-22 development: here are the [HTML] slides [1] detailing how effective Ada is in large projects

There's also this one from AdaCore. [2]

Looking at these things from an outside point of view, they look like language proselytization propaganda.

However, after writing a whole bunch of Ada code and then coming back to it multiple years later and seeing how easy things are to change, my skepticism has /considerably/ waned.

> language ahead of its time

It doesn't matter if you're sitting on a gold mine if no one knows about it. A massive problem is that Ada's vernacular predates or conflicts with a lot of modern terminology (storage_element, task, protected object, 'class isn't a "class", limited type, tagged instead of class, entry, "entry families", "controlled type" instead of "RAII", access instead of "handle" or "pointer" – yes, I know they're not exactly the same, "generic packages" – the closest thing might be an OCaml signature?). I tried to address some of these in my FOSDEM 2022 talk. [3]

[1] http://archive.adaic.com/docs/present/engle/whyada/tsld043.htm

[2] https://www.adacore.com/uploads/techPapers/Controlling-Costs-with-Software-Language-Choice-AdaCore-VDC-WP.PDF

[3] https://archive.fosdem.org/2022/schedule/event/ada_outsiders_guide/

Why Do You Keep Using Ada?

From: SabeDoesThings
Subject: Why Do You Keep Using Ada?
Date: Fri, 5 Sep 2025 12:01:05 +0000
Newsgroups: forum.ada-lang.io

I've been using Ada for over a month now and I've been really enjoying it and I wanted to connect more with the Ada community so I wanted to ask. How long have YOU been using Ada and what do

you like about it? I'm curious what people who have been using it way longer than me have to say about this awesome (and VERY underrated) language.

From: Nordic_Dogsledding
Date: Fri, 5 Sep 2025 15:20:08 +0000

I wrote avionics software for two helicopters in a big team.

Later I ported the original avionics (big endian, Ada 83) to a new compiler of a different vendor on different hardware (little endian, up to Ada 2012) for a real time flight training simulator. It was amazing how fast this was.

From: Irvise
Date: Sun, 7 Sep 2025 18:38:47 +0000

Disclaimer, I am not a professional programmer.

- Because it is fun to do things with it. I do not write code so that I can write code. I write code to do fun stuff and Ada just gets out of the way. Other languages are complex, require me to reason quite heavily, force me to think in the language-space rather than in my-problem-space.
- Because it is brutally easy to debug. I once found a stupid error in a library (un-initialised variable) in a matter of 20 mins. This is huge. The library was several thousands lines of code I had never seen. I only had to give it input and it gave me outputs. But with a specific, correct input, it crashed. I found the offending line in 15 mins thanks to Ada's check and arithmetic. Had it been any other language, it would have produced a garbage output and I would not have noticed it until several hours or days into my project (the output was not meant to be read by humans, it was automatically generated code).
- Because it made me learn a lot. I learnt more about low level code with Ada than C. I learnt more about types than any other languages.
- Because the community is super cool and knowledgeable.
- Because it is rock solid and well designed.
- Because it has features other langs can only dream about, such as type predicates, contracts and SPARK

I hope this answers your question.

From: jere
Date: Sun, 7 Sep 2025 19:42:56 +0000

I don't get to use Ada in my professional life as much (most things are done in C, but I try to sneak in some Ada when I can), so most of my Ada use these days is fun stuff at home.

For me it's just a fun language. I like most things about it (there's definitely some pain points for me, but they are minor), but I like how a well constructed package looks and feels. I like the type system, how enums are utilized, and the unique tasking system among other things.

From: pyj
Date: Sun, 7 Sep 2025 20:50:32 +0000

This is my personal list of why I learned the language in 2021ish and keep coming back to the language.

- Simple rules, clear syntax (easy to learn, and I've come back years later to a project and was able to pick it right up again).
- Ranges on types ('first, 'last, 'pred, 'succ, 'range).
- Super-powered enums ('first, 'last', 'pred, 'succ, 'value, 'image). Built-in to-string and parsing of enums, and you can use them as array indices.
- Pre/post conditions.
- Package instead of class-level encapsulation. Packages cut up subproblems at the domain level, not at some arbitrary software architecture level, so I get fewer getters/setters in tightly related code, fewer "God classes" [1], and keeping related types/functions together prevents ravioli code [2].
- Spec/implementation separation emphasizes "Why am I writing this?". Specs prevent the need to code fold and combine the public interface of a module and make it explicit.
- Good low level control (bit representation, compiler intrinsics, ASM, easy import from C)
- Ease of creating semantically different primitives (e.g. type Foo is new Integer). Helps with units and preventing data from flowing between systems without being cleaned.
- Named parameters and aggregates.
- Access types and aliased make "pointer" semantics much more direct.
- Operator overloading when I want it (i.e. math code).
- Explicit generics show the cost and slow down "generify all of the things" you see in other languages.
- Scope-based types (controlled types) for auto cleanup.
- Alire (95% of the time, I don't need to mess with the build).
- SPARK (opt-in verification).
- Mostly easy-to-use standard library.
- Environment startup task lets you explicitly control system startup before the program enters the entry point.

tldr;

Ada makes it easy to write the right thing the first time, even for low level code.

[1] https://en.wikipedia.org/wiki/God_object

[2] https://en.wikipedia.org/wiki/Spaghetti_code#Ravioli_code

From: pmnw
Date: Mon, 8 Sep 2025 09:11:17 +0000

We can enumerate the many benefits all we want, but I think it all boils down to Ada simply being an excellent tool of engineering.

From: kevlar700
Date: Thu, 11 Sep 2025 23:58:31 +0000

> We can enumerate the many benefits all we want, but I think it all boils down to

> Ada simply being an excellent tool of engineering.

And excellently engineered. It may have done some things differently with hind sight or maybe just today (UTF-8 dominance) but the language is all exhaustively well considered and engineered, which is rare and perhaps unique.

From: kevlar700
Date: Fri, 12 Sep 2025 00:15:39 +0000

I believe Dewar called it a "real marvel of engineering".

They set out to create the best language ever deserving to replace all other languages. Personally I think they achieved it. I guess the fact that the D.O.D. isn't saving as much money from using Ada as they should be is largely a social problem and not a technical one. Perhaps the NSA didn't actually want everyone to use a secure language too like they didn't want people using cryptography (Bernstein vs United States).

From: cup
Date: Mon, 8 Sep 2025 10:36:31 +0000

Because I'm a masochist. It takes me 3-5 times longer to write and debug an Ada program as it does C++/Fortran/Python/Java/C#/TCL/VBA. Sometimes, one syntax error can get me bogged down for half a day. Basically, I haven't rearranged the problem in my design to do it the Ada way. I'm still in C++/Python mode.

I keep using it because I'd like to think [in] Ada and speed up the development time instead of thinking in another language and then translating and having to rethink the problem and recode it the Ada way. It gets easier once I've worked out the whys and the hows.

From: JC001
Date: Tue, 9 Sep 2025 08:39:31 +0000

I like Ada because it supports the way I think as a software engineer.

I think all software should be correct. I use Ada because it is the best language for creating software that is correct, as shown by its long record of use in domains where the software must be correct.

My experience is that creating correct software with Ada takes less effort than creating incorrect software in other languages.

From: F-Loyer
Date: Tue, 9 Sep 2025 12:05:16 +0000

> My experience is that creating correct software with Ada takes less effort than creating incorrect software in other languages.

I feel the same. Strongly and statically typed language makes it harder to have a program which compiles… but once it compiles, it is often OK.

From: kevlar700
Date: Wed, 10 Sep 2025 08:30:56 +0000

Because it's the best language, better than C and Rust for embedded, better than Go for memory control and language design quality and reliability (Go shines at throwing people into short projects). Better than Rust at basically everything (I'm only not sure about concurrency performance but certainly concurrency safety on a standard runtime). I'm also now somewhat invested and whilst Zig looks interesting, I haven't actually investigated and I doubt it comes even close to what Ada SPARK offers.

From: tcoram
Date: Thu, 11 Sep 2025 17:50:57 +0000

I do a lot of embedded development, from real time industrial devices to 8-bit microcontrollers. It's the only language (outside of Forth on MCUs) that I don't have to *convince* I want to do low level embedded stuff.

And the tasking capabilities, warts et al, are still heads above any other programming languages.

Default_Component_Value with Subtype?

From: Blady
Subject: Default_component_value with Subtype?
Date: Sun, 28 Sep 2025 17:37:30 +0000
Newsgroups: forum.ada-lang.io

ARM 3.6 Array Types specifies: "Default_Component_Value shall be specified only on a full_type_declaration."

Thus, default initial values can only be given for types and not for subtypes. Is there any raison? This prevents the user from specifying a user specific default value on an extern library type which can't be changed. This also prevents specifying different default values for several subtypes.

From: jere
Date: Sun, 28 Sep 2025 23:02:29 +0000

The discussion the ARG had on it makes it sound like there was some sort of standard rule of thumb they followed in that decision, but I haven't looked through it long enough to see if I can find out why that rule of thumb existed

REF: http://www.ada-auth.org/cgi-bin/cvsweb.cgi/ai05s/ai05-0228-1.txt?rev=Top1.13

From: sttaft
Date: Mon, 29 Sep 2025 15:30:29 +0000

> The discussion the ARG had on it makes it sound like there was some sort of standard rule of thumb they followed in that decision

Actually, the AI explains its reasoning:

> We considered making these subtype aspects rather than type aspects. That would allow giving different default values to different subtypes. However, given the existence of many ways to create anonymous subtypes it would be very hard to guarantee that all values of a type are initialized. Without that guarantee, these aspects lose a lot of their purpose.

More generally, subtypes are somewhat "soft" in Ada, and particularly for scalar types, exactly which subtype is relevant in a given context can be somewhat unclear. As suggested, a derived type gives you much more control, and the /type/ of a scalar object is never in doubt.

Be that as it may, the aspect could probably be made to work with subtypes, but it would definitely add complexity to the typical Ada implementation.

From: Blady
Date: Mon, 29 Sep 2025 19:44:02 +0000

[...] It is surprising that I can give a Dynamic_Predicate aspect to a subtype but not Default_Component_Value!

From: sttaft
Date: Mon, 29 Sep 2025 21:17:02 +0000

> It is surprising that I can give a Dynamic_Predicate aspect to a subtype but not Default_Component_Value!

Good point. But predicates don't in general work very well if you have to put them on the "first" subtype (aka the type as a whole).

In fact, this combination was anticipated, but it seems that whatever compiler you are using didn't follow the rules stated in RM 3.2.4(31/5), which indicate a predicate is only checked on a default-initialized object if in fact the object has some explicitly specified default initialization:

> For an object created by an object_declaration [1] with no explicit initialization expression [2], or by an uninitialized allocator [3], if the types of any parts have specified Default_Value or Default_Component_Value aspects, or any subcomponents have default_expression [4]s, a check is performed that the value of the created object satisfies the predicates of the nominal subtype.

So you might want to file a bug report for this, if you have a small reproducer for this problem.

[1] https://ada-rapporteur-group.github.io/ARM/Ada_202Y/AA-3-3-1.html#S0032

[2] https://ada-rapporteur-group.github.io/ARM/Ada_202Y/AA-4-4.html#S0132

[3] https://ada-rapporteur-group.github.io/ARM/Ada_202Y/AA-4-8.html#S0164

[4] https://ada-rapporteur-group.github.io/ARM/Ada_202Y/AA-3-7.html#S0063

Ada Frontiers

Suggestions for Next Version of Ada Standard

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: Suggestions for next version of Ada standard
Date: Mon, 21 Jul 2025 20:19:28 +0000
Newsgroups: comp.lang.ada

[Distributed for the ARG —dc]

The Ada Rapporteur Group (ARG) of the ISO Working Group responsible for the Ada programming language standard (WG9) has a public GitHub repository for suggesting changes to the Ada standard:

https://github.com/Ada-Rapporteur-Group/User-Community-Input/issues/134

You can get there by starting at the ARG website: https://arg.adaic.org

We have also advertised on the Ada Forum: https://forum.ada-lang.io/

https://forum.ada-lang.io/t/plan-for-next-version-of-international-ada-standard/2007/1

Tucker Taft, on behalf of the Ada Rapporteur Group

New `finally` Keyword in GCC 15.1

From: Irvise
Subject: New `finally` Keyword in Gcc 15.1
Date: Mon, 28 Jul 2025 21:09:30 +0000
Newsgroups: forum.ada-lang.io

I failed to document (or maybe it was added a bit later than when I finished the changelog for GCC 15) that there is a new GNAT extension available in GNAT/GCC 15, it is the `finally` keyword. You can read about it in the GNAT Reference Manual [1]. There is not much documentation about it though nor examples. However, I came across this SPARK commit [2] which provided some examples. I tested the first one with a quick gnatmake -gnatX0 XXX.adb and it compiled without much fuss.

Sooo… how could this be used? I am asking as a complete newbie, not as a

critique. It is also nice to see that SPARK now supports it too!

[1] https://gcc.gnu.org/onlinedocs/gnat_rm/Finally-construct.html

[2] https://github.com/AdaCore/spark2014/commit/f2aaedcadeb71db2c61ba6706d5abcd50bbd6d6a

From: jere
Date: Tue, 29 Jul 2025 00:22:06 +0000

The first thought that comes to mind is something like files that you want to close if you successfully open them but later encounter an exception while parsing them. You can use the finally section to close the file regardless of whether the full process completed or exceptioned out.

From: dmitry-kazakov
Date: Wed, 30 Jul 2025 10:24:39 +0000

> how could this be used? I am asking as a complete newbie, not as a critique

It is a feature complement to object finalization. Finalization is an OO approach to the general problem of safe finalization. Finally is a procedural counterpart.

For example, let you have a mutex you seize to protect a critical code section. To ensure that the mutex gets released you can create a holder object that seizes the mutex on initialization and releases it on finalization. Then you place the holder in a scope like this:

```
declare
   Lock : Holder (Mutex'Access);
begin
   ... -- Critical section
end;
```

Now the procedural alternative would be:

```
Mutex.Seize;
begin
   ... -- Critical section
finally
   Mutex.Release;
end;
```

You can use it in smallish designs when procedural decomposition is sufficient.

Note also. You need to be very careful with the scope where you place /finalize/. In the example above it would be a bug to place Mutex.Seize after the /begin/. The compiler cannot protect you here.

From: sidisyom
Date: Sun, 3 Aug 2025 12:50:18 +0000

As mentioned, this may come in handy in cases where any type of resources acquired during the main sequence_of_statements need to be freed when the block exits (i.e. semaphores, files, memory etc).

One interesting point, as mentioned above, might be that the finally block gets executed whether the main sequence_of_statements terminates normally or with an exception. The RM mentions that: /*after the main sequence_of_statements is executed, and after any potential exception_handler is executed*/ but it's not explicitly mentioned whether that's the case for /unhandled/ exceptions too where no handlers exist in the exception block? Assume that's the case?

It also seems to be an /abort-deferred/ region when applying ATC.

From: Nordic_Dogsledding
Date: Sun, 3 Aug 2025 16:06:09 +0000

> The RM

Note: This is *not* the Ada RM. AdaCore are leaving the standard and beginning to develop an Ada derivation of their own.

> mentions that: /*after the main sequence_of_statements is executed, and after any potential exception_handler is executed*/

Now, what happens if the `finally` part raises an exception?

From: dmitry-kazakov
Date: Sun, 3 Aug 2025 16:14:17 +0000

> Now, what happens if the finally part raises an exception?

I guess the same when the exception part raises. It will propagate out of the block. This is sort of inconsistent with the claimed purpose of /finally/ should an unhandled exception happen in the middle of it. But what would you expect from Java-esque stuff? Alternatively one could nuke the program with Program_Error…

SPARK/Ada

[AdaCore Webinar] Introduction to Formal Verification with SPARK

From: Irvise
Subject: [adacore Webinar] Introduction to Formal Verification with Spark
Date: Mon, 14 Jul 2025 17:13:17 +0000
Newsgroups: forum.ada-lang.io

AdaCore just uploaded the recording of one of their webinars. I really liked it. It is a great summary and showcase of SPARK with real world examples and problems. Also, for those that were in AEiC 2025, this is closely related to Fabien's presentation (which I hope to edit and publish… soon?).

Anyway, here is the YT link for the video: https://youtu.be/tYAod_61ZuQ

And if someone has a good reputation/points/whatever_kids_have_nowadays in HackerNews and similar avenues, feel free to do so! A bunch of very interesting information is shared, specially regarding real world companies using Ada/SPARK for real world projects!

SPARK - Disabling Check after Proofs Are Ok

From: sbenitezb
Subject: Spark - Disabling Check after Proofs Are Ok
Date: Fri, 26 Sep 2025 17:13:26 +0000
Newsgroups: forum.ada-lang.io

After SPARK proofs pass, does the prover store this information and then the compiler use it to disable the runtime checks that are no longer needed? I couldn't find this information anywhere.

From: streaksu
Date: Fri, 26 Sep 2025 17:30:15 +0000

Nope, you need to disable it yourself, there is no GNAT gnatprove communication. Even if there was, you probably want to disable them anyways to support compilation without verification.

From: Irvise
Date: Sat, 27 Sep 2025 07:03:30 +0000

One way to potentially automate this is to create a shell script or Makefile which calls GNATprove && GNATmake without checks. If GNATprove errors out, GNATmake won't be called without checks. You can expand this idea so that if GNATprove fails, then GNATmake is called with checks, debug info, etc

From: JC001
Date: Sat, 27 Sep 2025 07:37:47 +0000

How would the ObjectAda compiler know about the results from the prover?

From: kevlar700
Date: Sat, 27 Sep 2025 17:30:02 +0000

There are also various pragmas such as suppress as some will want performance and some will want defense in depth or both in different places.

https://learn.adacore.com/courses/Ada_For_The_Embedded_C_Developer/chapters/05_SPARK.html

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.

2025

October 01-02 2025 **International Conference on Software Engineering Research & Development** (SERD'2025), Oklahoma City, Oklahoma, USA & Online. Topics include: general and social aspects of software engineering (SE); software design, testing, evolution, and maintenance; formal methods and theoretical foundations; programming languages (PLs), systems, and environments; object-oriented (OO) design and analysis; emerging SE technologies and dependability; distribution, componentization, and collaboration; concurrent, parallel, and distributed systems; etc.

October 06-10 25th **International Conference on Formal Methods in Computer-Aided Design** (FMCAD'2025). Menlo Park, California, USA. Topics include: formal aspects of computer-aided system design, including verification, specification, synthesis, and testing; groundbreaking methods, technologies, theoretical results, and tools for reasoning formally about computing systems; etc.

☺ October 12-18 ACM **Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2025), Singapore.

☺ Oct 12-25 **Conference on Object-Oriented Programming, Systems, Languages, and Applications** (OOPSLA'2025).

October 13-15 32nd **Static Analysis Symposium** (SAS'2025). Singapore. Co-located with SPLASH'2025. Topics include: static analysis as fundamental tool for program verification, bug detection, compiler optimization, program understanding, and software maintenance.

October 21-24 36th IEEE **International Symposium on Software Reliability Engineering** (ISSRE'2025), São Paulo, Brazil. Topics include: development, analysis methods and models throughout the software development lifecycle; dependability attributes (i.e., security, safety, maintainability, survivability, resilience, robustness) impacting software reliability; reliability threats, i.e. faults (defects, bugs, etc.), errors, failures; reliability means (fault prevention, fault removal, fault tolerance, fault forecasting); software testing and formal methods; software fault localization, debugging, root-cause analysis; reliability of AI-based systems; reliability of model-based and auto-generated software; reliability of open-source software; normative/regulatory/ethical spaces about software reliability; societal aspects of software reliability; etc.

October 21-24 17th **International Workshop on Software Aging and Rejuvenation** (WoSAR'2025). Topics include: progressive degradation of performability / availability / reliability / scalability / "-ilities" in software systems; analysis of aging-related faults/bugs, errors, and failures; software test strategies for detecting aging-related bugs; monitoring, detection, and data analysis of software aging effects (e.g., memory leaks, database index fragmentation, unterminated processes/threads, accrual of round-off errors); metrics for software performance and degradation; clone detection in aging software; tools for the detection and repair of memory leaks; etc.

October 27-30 23rd **Asian Symposium on Programming Languages and Systems** (APLAS'2025), Bangalore, India. Topics include: programming paradigms and styles (object-oriented programming, programming languages for systems code, ...), methods and tools to specify and reason about programs and languages (programming techniques, type systems, static and dynamic program analysis, language-based security,

model checking, testing, ...), programming language foundations, methods and tools for implementation (compilers, program transformations, refactoring, intermediate languages, run-time environments, garbage collection and memory management, build systems, ...), concurrency and distribution (concurrency theory, parallel programming, distributed computing, verification and testing of concurrent and distributed systems, ...), applications and emerging topics (programming languages and PL methods in education, security, privacy, signal processing, computer-aided design, ...; case studies in program analysis and verification). Deadline for submissions: October 10, 2025 (non-SRC posters).

October 27-31 23rd **International Symposium on Automated Technology for Verification and Analysis (**ATVA'2025). Bangalore, India. Topics include: theoretical and practical aspects of automated analysis, synthesis, and verification of hardware and software 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 of verification; automated tool support for verification; etc.

October 30 **Ada Switzerland** - half day Ada event. Schaffhausen, Switzerland.

November 05-06 18th **International Conference on Verification and Evaluation of Computer and Communication Systems** (VECoS'2025), Paris, France. Topics include: analysis of computer and communication systems, where functional and extra-functional properties are inter-related; cross-fertilization between various formal verification and evaluation approaches, methods and techniques, especially those developed for concurrent and distributed hardware/software systems.

☺ Nov 05-07 33rd **International Conference on Real-Time Networks and Systems** (RTNS'2025), Pisa, Italy. Topics include: real-time applications design and evaluation (automotive, avionics, space, railways, telecommunications, process control, ...), 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, ...), formal specification and verification, real-time distributed systems, etc. Deadline for early registration: October 15, 2025.

November 05 18th **Junior Researcher Workshop on Real-Time Computing** (JRWRTC'2025). Topics include: modeling, WCET, distributed systems, QoS, fault tolerance, scheduling, middleware, tools, applications such as automotive, avionics, telecom, and medical systems, etc. Deadline for submissions: October 3, 2025 (abstracts), October 8, 2025 (full papers).

November 10-13 26th **International Conference on Formal Engineering Methods** (ICFEM'2025), Hangzhou, China. Topics include: formal specification and modeling; formal approaches to fault prevention and detection; abstraction, refinement, and evolution; formal verification and validation; integration of formal methods and testing; practical formal methods; applications of formal methods; formal approaches to safety-critical system development; supporting tools for formal methods; formal methods for Cyber-physical systems; formal methods for security; formal certification of products; industrial case studies; etc.

November 10-14 23rd **International Conference on Software Engineering and Formal Methods** (SEFM'2025), Toledo, Spain. Topics include: aspects of software engineering and formal methods; software development methods (formal modelling, specification, and design; software evolution, maintenance, re-engineering, and reuse); design principles (programming languages; abstraction and refinement; ...); software testing, validation, and verification (model checking, theorem proving, ...; testing and runtime verification; other light-weight and scalable formal methods; ...); security and safety (security, privacy, and trust; safety-critical, fault-tolerant, and secure systems; software certification); applications and technology transfer (component, object, multi-agent systems; real-time, hybrid, and cyber-physical systems; intelligent systems and machine learning; education; ...); case studies, best practices, and experience reports.

☺ November 13 **High Integrity Software Conference** (HISC'2025), Newport, South Wales, UK. Topics include: The need for trustworthy software across a broader range of industries. Deadline for submissions: October 10, 2025 (posters).

☺ November 16 **Parallel Applications Workshop, Alternatives to MPI+X** (PAW-ATM'2025), St. Louis, Missouri, USA. Co-located with SC'2025. Topics include: alternatives to the MPI+X model, novel application

development using high-level parallel programming languages and frameworks; examples that demonstrate performance, compiler optimization, error checking, and reduced software complexity; experience with the use of new compilers and runtime environments; etc.

November 16-20 40th IEEE/ACM **International Conference on Automated Software Engineering** (ASE'2025), Seoul, South Korea. Topics include: foundations, techniques, and tools for automating analysis, design, implementation, testing, and maintenance of large software systems.

November 17-18 24th **Belgium-Netherlands Software Evolution Workshop** (BENEVOL'2025), Enschede, the Netherlands. Topics include: software evolution and maintenance. Deadline for author registration: November 1, 2025.

November 19-21 20th **International Conference on integrated Formal Methods** (iFM'2025), Paris, France. Topics include: recent research advances in the development of integrated approaches to formal modelling and analysis; all aspects of the design of integrated techniques, including language design, verification and validation, automated tool support and the use of such techniques in software engineering practice.

November 24-28 22nd **International Colloquium on Theoretical Aspects of Computing** (ICTAC'2025), Marrakech, Morocco. Topics include: semantics of programming languages; theories of concurrency; theories of distributed computing; models of objects and components; security; static analysis; software verification; timed, hybrid, and cyber-physical systems; software testing; model checking and theorem-proving; applications and experience; etc.

☺ November 26-28 6th **International Conference on Reliability, Safety and Security of Railway Systems** (RSSRail'2025), Pisa, Italy. Topics include: safety in development processes and safety management; combined approaches to safety and security; system and software safety analysis; formal modelling and verification techniques; system reliability; validation according to the standards; tool and model integration, tool chain; domain-specific languages and modelling frameworks; model reuse for reliability, safety and security; etc.

November 26-28 9th **International Conference on System Reliability and Safety** (ICSRS'2025), Turin, Italy. Topics include: system design and modeling based on reliability, engineering reliability maintenance and safety management, etc.

December 01-03 11th **International Symposium on Dependable Software Engineering: Theories, Tools and Applications** (SETTA'2025). Oxford, UK. Topics include: formalisms for modeling, design and implementation; model checking, theorem proving, and decision procedures; scalable approaches to formal system analysis; formal approaches to simulation, run-time verification, and testing; integration of formal methods into software engineering practice; contract-based engineering of components, systems, and systems of systems; formal and engineering aspects of software evolution and maintenance; parallel and multicore programming; embedded, real-time, hybrid, probabilistic, and cyber-physical systems; mixed-critical applications and systems; safety, reliability, robustness, and fault-tolerance; dependability of smart software and systems; applications and industrial experience reports; software tools to assist the construction or analysis of software systems; etc.

☺ Dec 02-05 46th IEEE **Real-Time Systems Symposium** (RTSS'2025), Boston, Massachusetts, USA. Topics include: innovations covering all aspects of real-time systems, including theory, design, analysis, implementation, verification, evaluation, and experience. Deadline for submissions: October 8, 2025 (Brief Presentations track).

December 02-05 28th **Brazilian Symposium on Formal Methods** (SBMF'2025). Recife, Pernambuco, Brazil. Topics include: development, dissemination, and use of formal methods for construction of high-quality computational systems; applications of formal methods to software design, software development, software code generation, software testing, software maintenance, evolution and reuse, ...; specification and modeling languages (logic and semantics for specification and programming languages; formal methods for timed, real-time, hybrid, or safety-critical systems; formal methods for cyber-physical systems; ...); verification and validation (abstraction, modularization, and refinement techniques; static analysis; model checking; theorem proving; software certification; correctness by construction); experience reports (reports on teaching formal methods, reports on the industrial application of formal methods); etc. Deadline for early registration: November 9, 2025.

December 02-05 32nd **Asia-Pacific Software Engineering Conference** (APSEC'2025), Macao SAR, China. Topics include: requirements and design (component-based software engineering; software architecture,

modeling and design; middleware, frameworks, and APIs; software product-line engineering; ...); testing and analysis (testing, verification, and validation; program analysis; program repairs; ...); formal aspects of software engineering (formal methods; model-driven and domain-specific engineering); software comprehension and traceability; dependability, safety, and reliability; software maintenance and evolution (refactoring; reverse engineering; software reuse; debugging, defect prediction, and fault localization; software repository mining; ...); etc.

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

December 15-19 45th **Conference on Foundations of Software Technology and Theoretical Computer Science** (FSTTCS'2025), BITS Pilani, K K Birla Goa Campus, India. Topics include: formal methods; principles and semantics of programming languages; program analysis and transformation; specification, verification and synthesis; theorem proving; etc.

Dec 17-19 **Workshop on Research Highlights in Programming Languages** (RHPL'2025). Topics include: applied formal methods, compilers, concurrency and memory models, program analysis and verification, programming language design and semantics, etc.

2026

☺ January 11-17 53rd ACM SIGPLAN **Symposium on Principles of Programming Languages** (POPL'2026). Rennes, France. Topics include: all aspects of programming languages and programming systems, both theoretical and practical, ranging from formal frameworks to experience reports; principled, enduring contributions to the theory, design, understanding, implementation, or application of programming languages. Deadline for submissions: October 10, 2025 (tutorials).

January 12-13 **International Conference on Certified Programs and Proofs** (CPP'2026). Topics include: research areas related to formal certification of programs and proofs; new languages and tools for certified programming; program analysis, program verification, and program synthesis; program logics, type systems, and semantics for certified code; verification of correctness and security properties; teaching mathematics and computer science with proof assistants; etc.

January 12-13 27th **International Conference on Verification, Model Checking, and Abstract Interpretation** (VMCAI'2026). Rennes, France. Co-located with POPL'2026. Topics include: program verification, model checking, abstract interpretation, static analysis, type systems, program certification, detection of bugs and security vulnerabilities, hybrid and cyber-physical systems, concurrent and distributed systems, analysis of numerical properties, analysis of smart contracts, etc., case studies on all of the above topics.

January 26-28 21st **International Conference on High Performance and Embedded Architecture and Compilation** (HiPEAC'2026), Kraków, Poland. Topics include: computer architecture, programming models, compilers and operating systems for general-purpose, embedded and cyber-physical systems. Areas include safety-critical dependencies, cybersecurity, energy efficiency and machine learning. Deadline for submissions: November 4, 2025 (TechNexus Programme workshops).

Jan 31 – Feb 01 35th ACM SIGPLAN **International Conference on Compiler Construction** (CC'2026), Sydney, Australia. 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; design and implementation of novel language constructs, programming models, and domain-specific languages. techniques, ranging from programming languages to micro-architectural support, for various domains of computing, including secure, parallel, distributed, embedded, or mobile environments; programming tools, including refactoring editors, checkers, verifiers, compilers, debuggers, and profilers; etc. Deadline for submissions: November 7, 2025 (abstracts), November 10, 2025 (full papers).

Jan 31 – Feb 04 IEEE/ACM **International Symposium on Code Generation and Optimization** (CGO'2026). Sydney, Australia. Topics include: a wide range of optimization and code generation techniques and related issues at the interface of hardware and software.

March 17-20 33rd IEEE **International Conference on Software Analysis, Evolution, and Reengineering** (SANER'2026). Limassol, Cyprus. Topics include: software analysis, parsing, and fact extraction; software maintenance and evolution, evolution analysis; software reverse engineering and reengineering; program comprehension; software architecture recovery and reverse architecting; program transformation and refactoring; mining software repositories and software analytics; software reconstruction and migration; program repair; software release engineering, continuous integration and delivery; software security, safety, reliability and quality analysis; software tools for software evolution and maintenance; empirical studies in the context of software analysis, evolution and reengineering; education and training in the context of software analysis, evolution and reengineering; etc. Deadline for submissions: October 9, 2025 (research track abstracts), October 16, 2025 (research track papers), November 7, 2025 (registered report track), November 10, 2025 (industrial track abstracts, short papers and posters track abstracts, Early Research Achievement (ERA) track abstracts, Reproducibility Studies and Negative Results (RENE) track abstracts), November 17, 2025 (industrial track papers, short papers and posters track papers, Early Research Achievement (ERA) track papers, Reproducibility Studies and Negative Results (RENE) track papers, tool demo track papers), December 8, 2025 (Journal-First track papers), December 12, 2025 (workshop abstracts), December 18, 2025 (workshop papers), January 12, 2026 (tutorials).

March 23-26 32nd **International Working Conference on Requirements Engineering: Foundation for Software Quality** (REFSQ'2026), Poznan, Poland. Theme: "Trustworthy and Ethical Systems via Requirements Engineering". Deadline for submissions: October 6, 2025 (workshops), October 17, 2025 (research track papers), January 6, 2026 (education and training papers), January 19, 2026 (posters & tool papers).

March 23-27 41st ACM/SIGAPP **Symposium on Applied Computing** (SAC'2026), Thessaloniki, Greece.

☺ Mar 23-27 **Track on Programming Languages** (PL'2026). Topics include: technical ideas and experiences relating to implementation and application of programming languages, such as compiling techniques, domain-specific languages, formal semantics, 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. Deadline for submissions: October 17, 2025 (full papers). Deadline for author registration: December 12, 2025.

March 23-27 **Software Verification and Testing Track** (SVT'2026). 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. Deadline for submissions: October 17, 2025 (regular papers, student research competition research abstracts). Deadline for author registration: December 12, 2025.

March 23-27 21st **Track on Dependable, Adaptive, and Secure Distributed Systems** (DADS'2026). Topics include: Dependable, Adaptive, and secure Distributed Systems (DADS); modeling, design, and engineering of DADS; foundations and formal methods for DADS; applications of DADS; etc.

April 07-10 21st **European Dependable Computing Conference** (EDCC'2026), Canterbury, UK. Topics include: latest research results on theory, techniques, systems, and tools for design, validation, operation, and evaluation of dependable and secure computing systems; hardware and software architecture of dependable systems; dependability and security modelling, evaluation, benchmarking, and tools; safety-critical systems design and analysis; mixed-criticality systems design and evaluation; testing and validation methods; dependability and security of artificial intelligence systems, cyber-physical systems, e.g., smart grids, intelligent vehicles, (industrial) Internet of Things; ...; etc. Deadline for submissions: October 10, 2025 (paper abstracts, workshops), October 17, 2025 (full papers), January 27, 2026 (extended & fast abstracts, student forum papers), February 22, 2026 (posters).

April 11-16 29th ETAPS **International Joint Conferences on Theory and Practice of Software** (ETAPS'2026), Torino, Italy. Deadline for submissions: October 16, 2025 (ESOP round 2, TACAS, FoSSaCS, FASE), October 30, 2025 (TACAS artifacts), January 8, 2026 (non-TACAS artifacts).

April 13-16 35th **European Symposium on Programming** (ESOP'2026) Topics include: fundamental issues in the specification, design, analysis, and implementation of programming languages and systems, such as programming paradigms and styles, methods and tools to specify and reason about programs and languages, programming language foundations, methods and tools for implementation, concurrency and distribution, etc. Deadline for submissions: October 16, 2025 (round 2).

April 13-16 29th **International Conference on Foundations of Software Science and Computation Structures** (FoSSaCS'2026). Topics include: theories and methods to support analysis, integration, synthesis, transformation, and verification of programs and software systems; language theory, automata, and games; type theory and proof theory; concurrency theory and process calculi; semantics of programming languages; program analysis, correctness, transformation, and verification; logics of programming; software specification and refinement; models of software security; etc. Deadline for submissions: October 12, 2025 (papers), January 4, 2026 (artefacts).

April 12-18 48th IEEE/ACM **International Conference on Software Engineering** (ICSE'2026), Rio de Janeiro, Brazil. Topics include: mining software repositories; software design methodologies, principles, and strategies; architecture quality attributes, such as security, privacy, performance, reliability; modularity and reusability; dependency and complexity analysis; patterns and anti-patterns; technical debt in design and architecture; formal methods and model checking; reliability, availability, and safety; resilience and antifragility; design for dependability and security; vulnerability detection to enhance software security; evolution and maintenance; software reuse; refactoring; environments and software development tools; focusing on programming languages, environments, and tools supporting individuals, teams, communities, and companies; systems and software traceability; modeling languages, techniques, and tools; empirical studies on the application of model-based engineering; software testing; automated test generation techniques such as fuzzing, search-based approaches, and symbolic execution; program analysis; debugging and fault localization; runtime analysis and/or error recovery; etc. Deadline for submissions: October 6, 2025 (SIGSOFT award nomination intentions), October 20, 2025 (workshop papers, SIGSOFT award nominations).

April 12-13 14th **International Conference on Formal Methods in Software Engineering** (FormaliSE'2026). Topics include: approaches, methods, and tools for verification and validation; formal approaches to safety and security-related issues; scalability of formal method applications; integration of formal methods within the software development lifecycle; model-based engineering approaches; correctness-by-construction approaches for software and systems engineering; application of formal methods to specific domains, e.g., autonomous, cyber-physical, intelligent, and IoT systems; formal methods in a certification context; case studies developed/analyzed with formal approaches; experience reports on the application of formal methods to real-world problems; guidelines to use formal methods in practice; usability of formal methods; etc. Deadline for submissions: November 6, 2025.

May 05-07 18th **NASA Formal Methods Symposium** (NFM'2026), Los Angeles, California, USA. Topics include: identifying challenges and providing solutions towards achieving assurance for critical systems; formal techniques for software and system assurance for applications in space, aviation, robotics, and other NASA-relevant safety-critical systems. Deadline for submissions: January 10, 2026 (papers).

May 11-14 19th **Cyber-Physical Systems and Internet of Things Week** (CPS-IoT Week'2026), Saint Malo, France. Event includes: 4 top conferences, HSCC, ICCPS, RTAS, and SenSys, as well as multiple workshops, tutorials, competitions, and various exhibitions from both industry and academia. Topics include: embedded & real-time systems, system reliability and security, etc. Deadline for submissions: November 6, 2025 (abstracts), November 13, 2025 (papers), November 20, 2025 (workshops, tutorials, competitions).

☺ May 12-14 32nd IEEE **Real-Time and Embedded Technology and Applications Symposium** (RTAS'2026). Topics include: applications with timing requirements; real-time and embedded operating systems; application profiling, WCET analysis, compilers, tools, benchmarks and case studies; modelling languages, modelling methods, model validation and calibration; scheduling and resource allocation; verification and validation methodologies; etc. Deadline for submissions: November 13, 2025 (papers), February 13, 2026 (brief presentations).

May 19-21 — 18th **Software Quality Days** (SWQD'2026), Vienna, Austria. Theme: "Software Architecture as Backbone of Software Quality". Topics include: software and systems quality; methods and tools for constructive and analytical quality assurance; testing of software-intensive and AI-based systems; process improvement for development and testing; automation in quality assurance and testing; software architecture and design; continuous integration, deployment, and delivery (CI/CD); software modernization and evolution; technical debt and quality degradation over time; secure coding and security by design; case studies and experience reports on practical applications; etc. Deadline for submissions: October 31, 2025 (papers).

May 20-22 — 27th **International Symposium on Formal Methods** (FM'2026), Tokyo, Japan. Topics include: development and application of formal methods in a wide range of domains including trustworthy AI, computer-based systems, systems-of-systems, cyber-physical systems, security, human-computer interaction, manufacturing, sustainability, energy, transport, smart cities, healthcare and biology; techniques, tools, and experiences in interdisciplinary settings; experiences of applying formal methods in industrial settings; design and validation of formal method tools; etc. Deadline for submissions: October 26, 2025 (workshops), November 25, 2025 (abstracts), December 2, 2025 (full papers), February 2, 2026 (tutorials).

♦ June 09-12 — 30th **Ada-Europe International Conference on Reliable Software Technologies** (AEiC'2026). Västerås, Sweden. Organized by Ada-Europe and Mälardalen University (MDU). Deadline for submissions: January 16, 2026 (journal track papers), February 27, 2026 (regular and industrial track papers, work-in-progress papers, tutorial and workshop proposals). Submit early: acceptance decisions for the journal track and workshops are made on a rolling basis! #AEiC2026 #AdaEurope #AdaProgramming

June 22-26 — 23rd IEEE **International Conference on Software Architecture** (ICSA'2026), Amsterdam, the Netherlands. Deadline for submissions: October 3, 2025 (workshops), November 28, 2025 (Research Track paper abstracts, Software Architecture in Practice Track paper abstracts), December 5, 2025 (Research Track papers, Software Architecture in Practice Track papers).

☺ Jun 29 – Jul 03 — 40th **European COnference On Programming** (ECOOP'2026), Brussels, Belgium. Formerly: European Conference on Object-Oriented Programming. Topics include: all topics related to programming languages, software development, systems and applications; all practical and theoretical investigations of programming languages, systems and environments; innovative solutions to real problems as well as evaluations of existing solutions. Deadline for submissions: November 27, 2025 (round 1), February 12, 2026 (round 2).

Jun 29 – Jul 03 — **Software Technologies: Applications and Foundations** (STAF'2026), Rennes, France. Topics include: all aspects of software technologies and languages, including object-oriented design, testing, formal modeling and verification, model transformation, model-driven and aspect-oriented approaches, and related tools. Deadline for submissions: December 8, 2025 (workshops).

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



General Chairs

Cristina Seceleanu, *Mälardalen University*
cristina.seceleanu@mdu.se

Kristina Lundqvist, *Mälardalen University*
kristina.lundqvist@mdu.se

Journal track Co-chairs

Axel Jantsch, *TU Wien*
axel.jantsch@tuwien.ac.at

António Casimiro, *University of Lisbon*
casim@ciencias.ulisboa.pt

Industrial track Co-chairs

Sahar Tahvili, *Einride AB*
sahar.tahvili@einride.tech

Rong Gu, *Mälardalen University*
rong.gu@mdu.se

Regular papers track Co-chairs

Antonio Filieri, *Amazon Web Services*
afilieri@amazon.com

Peter Backeman, *Mälardalen University*
peter.backeman@mdu.se

Work-in-progress track Co-chairs

Malvina Latifaj, *Mälardalen University*
malvina.latifaj@mdu.se

Federico Aromolo, *Scuola Sant'Anna*
federico.aromolo@santannapisa.it

Workshop Chair

Dene Brown, *SysAda Limited*
dene.brown@sysada.co.uk

Tutorial Chair

Henry Muccini, *University of L'Aquila*
henry.muccini@univaq.it

Exhibition & Sponsorship Chair

Tiberiu Seceleanu, *Mälardalen University*
tiberiu.seceleanu@mdu.se

Finance Chair

Ahlan Marriott, *White Elephant GmbH*
ahlan@ada-switzerland.ch

Publicity Chair

Dirk Craeynest, *Ada-Belgium & KU Leuven*
dirk.craeynest@cs.kuleuven.be

Webmasters

Hai Nam Tran, *University of Brest*
hai-nam.tran@univ-brest.fr

Leo Hatvani, *Mälardalen University*
leo.hatvani@mdu.se

Local Chair

Susanne Frønnå, *Mälardalen University*
susanne.fronna@mdu.se

General Information

The **30th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2026)** will take place in Västerås, Sweden. The conference schedule comprises keynote talks, a journal track, an industrial track, a regular papers track, a work-in-progress track, a vendor exhibition, parallel tutorials, and satellite workshops.

- Journal track papers present journal-level research advances supported by solid theoretical foundation and thorough evaluation.
- Industrial track contributions highlight the practitioners' side of a challenging case study or industrial project.
- Regular track papers present conference-level (full paper) research advances.
- Work-in-progress track papers illustrate novel research ideas that are still at an initial stage, between conception and first prototype.
- Tutorials guide attenders through a hands-on familiarization with innovative developments or with useful features related to reliable software.
- Workshops provide discussion forums on the conference topics.
- Companies presentations and exhibitions to showcase products and services.

Schedule

16 January 2026	Deadline for submission of journal track papers
27 February 2026	Deadline for submission of industrial track abstracts, regular track papers, work-in-progress papers, and tutorial and workshop proposals
27 March 2026	First round notification for journal track papers, and notification of acceptance for all other types of submissions
9-12 June 2026	Conference

Scope and 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 allow ample time for keynotes, Q&A sessions, discussions, and 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:

- Formal methods, testing, and model-based engineering of critical systems
- High-integrity systems and reliability
- AI for safety-critical and high-integrity systems engineering
- Real-time systems
- Ada language and technologies
- Applications and experimental evaluations in relevant domains

More specific topics are described on the conference web page.

Call for Journal Track Submissions

The conference includes a journal track, which seeks original and high-quality papers, not submitted elsewhere, which describe mature research work on the conference topics. Accepted journal track papers will be published in a Special Issue of Elsevier's JSA (www.sciencedirect.com/journal/journal-of-systems-architecture/) – the *Journal of Systems Architecture* (Q1 ranked, CiteScore 10.5, impact factor 4.1). Contributions must be submitted by **16 January 2026** and should not exceed **25 pages** (in case the authors need more pages the journal track chairs need to be contacted)**.** JSA has adopted the Virtual Special Issue model to speed up the publication process, where Special Issue (SI) papers are published in regular issues but marked as SI papers. Acceptance decisions are made on a rolling basis. Therefore, authors are encouraged to submit papers early and need not wait until the submission deadline. Authors who have successfully passed the first round of review (including major revisions) will be invited to present their work at the conference. Prospective authors may direct all enquiries regarding this track to the corresponding chairs.

Call for Regular Track Submissions

The conference includes a regular track, which seeks original and high-quality submissions, not submitted elsewhere, which describe mature research work on the conference topics. Accepted papers will be published in the *Open Access Series in Informatics* (OASICS series: drops.dagstuhl.de/entities/series/OASIcs) and should be formatted according to the one-column format, not exceeding **16 pages**, excluding references. Contributions must be submitted by **27 February 2026.**

Call for Industrial Track Submissions

The conference seeks industrial-practitioner presentations that deliver insight on the challenges of developing reliable software. Especially welcome kinds of submissions are listed on the conference web site. Given their applied nature, such contributions will be subject to a dedicated practitioner-peer-review process. Interested authors shall submit a **one-to-two pages** abstract, by **27 February 2026**. Authors will also be invited to expand their contributions into full-fledged articles for publication in the *Ada User Journal* (www.ada-europe.org/auj/scope/), which will form the proceedings of the industrial track of the Conference.

Call for Work-in-progress Track Submissions

The work-in-progress (WiP) track seeks two kinds of submissions: (a) ongoing research and (b) early-stage ideas. Ongoing research submissions are **4-page** papers describing research results that are not mature enough to be submitted to the journal or regular tracks. Early-stage ideas are **1-page** papers that pitch new research directions that fall within the scope of the conference. Both kinds of submissions must be original and shall undergo anonymous peer review. Submissions by recent M.Sc. graduates and Ph.D. students are especially sought. Authors shall submit their work by **27 February 2026**. Authors will also be offered the opportunity to publish their contributions (and expand to a 4-page article for the 1-papers) in the *Ada User Journal* (www.ada-europe.org/auj/scope/), which will form the proceedings of the WiP track of the conference.

Call for Tutorials

The conference seeks tutorials in the form of educational seminars on themes falling within the conference scope, with an academic-for-practitioner slant, including hands-on or practical elements. 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), the intended level of the contents (introductory, intermediate, or advanced), and a statement motivating attendance. Tutorial proposals shall be submitted at any time but no later than **27 February 2026** to the respective chairs. The authors of accepted full-day tutorials will receive a complimentary conference registration, halved for half-day tutorials. The *Ada User Journal* will offer space for the publication of summaries of the accepted tutorials.

Call for Workshops

The conference welcomes satellite workshops centred on themes that fall within the conference scope. Proposals may be submitted for half- or full-day events, to be scheduled at either end of the AEiC conference. Workshop organizers shall also commit to producing the proceedings of the event, for publication in the *Ada User Journal*. Workshop proposals shall be submitted at any time but no later than **27 February 2026** to the respective chair. Once submitted, each workshop proposal will be evaluated by the conference organizers as soon as possible.

Call for Exhibitors

The conference will include a vendor and technology exhibition. Interested providers should direct inquiries to the Exhibition & Sponsorship Chair.

Venue



The conference is in-person, and will take place at Mälardalen University, Västerås, Sweden. Mälardalen University (MDU), one of Sweden's youngest and most dynamic universities, is a leading institution for applied research and innovation in collaboration with industry and the public sector. Established in 1977 and granted full university status in 2022, MDU is home to over 17,000 students and 1,000 staff members, with strong international ties and research excellence in areas such as embedded systems, future energy, health and welfare, and sustainable development. The university houses several internationally recognized research environments, out of which Embedded Systems, Innovation and Product Realization, and Future Energy are among the most prominent ones. MDU's campuses (Västerås and Eskilstuna) are modern, centrally located, and purpose-built to foster interdisciplinary collaboration and innovation.

Västerås is one of Sweden's oldest cities and now a hub for advanced technology, energy innovation, and digital transformation. Strategically located just an hour west of Stockholm and easily accessible by train, bus or car, Västerås is home to global tech companies such as ABB, Alstom, Westinghouse. The city sits on the shores of Lake Mälaren and combines scenic beauty, a vibrant cultural life, and excellent infrastructure. Västerås Airport offers direct flights to several European destinations, and Stockholm Arlanda International Airport is reachable within 90 minutes. With its blend of historical charm, green parks, and cutting-edge industry, Västerås offers the perfect setting for an inspiring and high-impact academic and industrial gathering.

Automated Execution of Attack Chains to Train Data-Driven IDS in Cyber-physical Systems

Diogo Pedro, José Cecílio, Pedro M. Ferreira, Alan Oliveira de Sá
LASIGE, Faculdade de Ciências, Universidade de Lisboa, Lisboa, Portugal; email: fc56337@alunos.fc.ul.pt, {jmcecilio, pmf, aodsa}@ciencias.ulisboa.pt

Abstract

The rapid expansion of Cyber-Physical Systems (CPS), their extensiveness, and their wide range of applications have introduced several new security challenges, particularly in critical infrastructures such as smart grids or vehicles. The growing attack techniques, diversity of devices, and network technologies are becoming more sophisticated, making the infrastructure more vulnerable. Meanwhile, machine learning-based intrusion detection systems (IDS) for CPS have struggled to keep up due to the lack of datasets and limited variability in attack types and system architectures. Therefore, it is essential to improve the diversity, quality, and representativeness of datasets used to train CPS IDSs. This work aims to present a framework capable of automatically execute attack chains in simulated CPS environments, producing datasets to aid the development of effective IDSs. The framework leverages an ontology that allows the tool to interpret attack paths and be easily extensible.

Keywords: Cyber-Physical Systems, Cyberattacks, Honeynet, Attack Automation, Semantic modeling, Ontology.

1 Introduction

A cyber-physical system (CPS) comprises physical and cyber processes, coordinating the two worlds, enabling software to monitor and act upon physical processes. These systems are the foundation of our everyday technological society, enabling the production of millions of products in factories, efficient electricity distribution with smart grids, smart vehicles, and much more [1].

An important subset of CPS is Industrial Control Systems (ICS), which manage industrial processes through actuators, sensors, and control systems such as Supervisory Control and Data Acquisition (SCADA). These SCADA systems include several types of controllers, such as Programmable Logic Controllers (PLCs), Remote Terminal Units (RTUs), and Human Machine Interfaces (HMIs) for monitoring and visualization.

The number of Cyberattacks has been high among CPS, especially with the rise of connected devices from the evolution of Industry 4.0 and the Internet of Things (IoT) [2]. To aggravate, threats have grown more sophisticated, with highly organized groups developing Advanced Persistent Threats (APTs) such as Stuxnet [3] and BlackEnergy [4], two examples that have caused significant damage. Honeynets are used to gather attacker information and waste their time on simulated systems [5]. Others such as Intrusion Detection Systems (IDS) are also critical, monitoring network traffic to identify, alert, and possibly mitigate malicious activities. In the CPS scope, these tools and techniques have struggled to keep up with non-CPS counterparts, hence the need for efforts to promote more secure cyber-physical environments.
Many IDSs work with policies and logic rules to identify those activities. Modern IDSs increasingly rely on machine learning (ML), but its effectiveness depends on extensive datasets of real-world CPS attacks. However, datasets for CPS are limited, often lacking diversity in attack types or system configurations that do not represent the extensive variety of possible architectural configurations as exposed by Martins et al. [6]. As their work presents, the gap arises from the sensitivity of the data and the diversity of protocols that cause difficulties in gathering the information coherently. Therefore, it is essential to have tools capable of producing datasets with the necessary diversity.
The framework proposed in this work aims to execute random attack chains in simulated environments with configurable settings, allowing for testing without hardware and without sharing sensitive information. It also collects network data and system logs, as discussed in [6], to support the development of more effective machine learning-based intrusion detection systems (IDSs).
Sousa et al. [7] developed a simulated Honeynet environment that can be configured in any desired architecture. Its containerized and modular application, that accrues from their docker implementation, ensures high scalability, enabling the inclusion of various systems within the dataset. Building upon this work, we aim to further enhance and expand its capabilities, by extending its architecture to automatically compile and execute attack chains that enable gathering diversified datasets for training ML-based IDSs for CPS.
For that purpose, we design a structured ontology approach to describe attacks and system components, facilitating the interpretation of attack paths and enabling the system's extensibility. Lastly, we aim at the design and implementation of an attack coordinator component that automatically executes attack chains, allowing the simulation of different attack scenarios and the collection of data over those scenarios.
The remaining of this paper is structured as follows: Section 2 discusses valuable works that serve as the foundational research for this study. Section 3 covers the topics of the system design, modeling, and attack coordinator's components, while the last section concludes with the forthcoming work.

2 Related works

This Section explores the current state-of-the-art topics related to this work, including an overview of the studied attacks

Attack	OT/IT	Attack	Target	Protocol	System	Tool	CVE	CWE	CVSS	Mitigation
Ping sweep	IT	Access to network	Router	ICMP	-	Nmap [1]	-	200	-	Firewall
Port Scan	IT	Access to network	Router	TCP, UDP	-	Nmap [1]	-	200	-	Firewall, Ports Protection
Remote command execution (RCE) in FUXA endpoint	OT	FUXA 1.1.13	HMI	HTTP	FUXA	Script [2]	2023-33831	77	9.8	-
Command Injection in Open PLC Webserver v3	OT	OpenPLC WebServer v3	PLC	HTTP(s)	OpenPLC	Script [3]	2021-31630	94	8.8	Stricter input validation
Register Scan	OT	Network information of devices using Modbus/TCP	Modbus/ TCP	Modbus/ TCP	Communi-cation of two devices using Modbus/TCP	Script [4]	-	200	-	Setup Firewall

Table 1: Attacks and vulnerabilities selected to be explored in this work.

against CPSs, attack chains, and methods used for attack automation. It also provides an overview of the current datasets and IDSs used in CPS.

2.1 Attacks

Humayed et al. [2] present comprehensive research on CPS security, providing a descriptive taxonomy that categorizes vulnerabilities, threats, and defense mechanisms across cyber, physical, and cyber-physical layers. Their research highlights critical vulnerabilities in ICS, such as insecure communication protocols, lack of authentication, and sensor spoofing. According to their work, several attacks present a low complexity but high impact mainly Man-in-the-Middle, Denial of Service, Replay Attacks, Modbus-Register Reading, and Spoofing. Given their context this should be some of the first to be included in this framework. One of the insecure communication protocols mentioned is Modbus/TCP, which is still widely used across many CPS. Despite its prevalence, its lack of encryption and authentication makes it susceptible to Flooding, Man-in-the-Middle, and Replay attacks [8, 9, 10]. These vulnerabilities make Advanced Persistent Threats (APTs) even more prevalent.

In addition to the works referenced, this research also analyzed vulnerability databases like CVEdetails, Snyk, MITRE CVE, and NVD NIST, focusing on PLCs such as OpenPLC[5] and HMIs like FUXA[6]. Table 1 summarizes examples of attacks and vulnerabilities found in the literature that are explored in this work, in both Operational Technology (OT) and Information Technology (IT) scopes. The collected information comprises the attack name, description, requirements, and targets. Complementarily, references to the exploits, mitigation, known identifiers (CVE, CWE), and CVSS scores (which indicates vulnerability severity) are also included.

2.2 Attack chain automation

Automating execution of attacks has long been a focus in cybersecurity, with many attempts along the way and important progress shown with tools like fuzzers, and others that leverage techniques such as attack trees [11], attack graphs [12] and knowledge graphs [13]. These methods enable modeling, analysis, and simulation of attack paths within networks and systems, helping to identify and mitigate threats.

Attack graphs have been extensively used in the literature to model and automate attacks by outlining possible exploitation paths. For instance, Al Ghazo et al. [14] demonstrate their application in SCADA networks with a methodology for automatic attack graph generation and visualization, enhancing security monitoring and analysis.

Recent advancements integrate cybersecurity knowledge graphs to enhance attack graph generation. As in the work proposed by Qi et al. [15], a knowledge graph framework enhances automated detection and creation of attack graphs to improve accuracy and efficiency in both CPS and enterprise systems. Others use knowledge graph techniques to represent attack relationships, enhancing accuracy and human understanding of the automated generation of attack graphs [16].

OpenPLCReinforcement learning (RL) and ML are also being explored for automated penetration testing. Despite the progress made, their applicability to CPS remains limited due to the scarcity of datasets capturing the unique characteristics of the interplay between physical and digital components and the diversity of architectures and system components. Addressing this gap remains a critical challenge for advancing the use of RL and related techniques in securing CPS.

Compared to the literature discussed in this section, this work aims to facilitate the execution of attack chains within a high-interaction CPS environment in order to provide realistic datasets for training ML-based IDS. Note that it includes not only optimized attack paths, typically performed by highly skilled attackers/algorithms, but also others that could be taken by less experienced attackers/algorithms.

3 Automation of attack chains

This work builds on Sousa et al. [7], retaining the concept of a configurable simulated environment while focusing on developing the Attack Coordinator module. Figure 1 shows an example system architecture, where all attacks target the simulated environment. The key advantage of the proposed work is the ability to generate a variety of system configurations via the Architecture Coordinator module. So far, the possible architectural configurations comprise different arrangements of components like HMI, PLCs, and simulated plants.

The Plant simulates a physical process, communicating with the PLC via Modbus/TCP and responding to its commands.

[1] https://nmap.org/book/toc.html
[2] https://github.com/rodolfomarianocy/Unauthenticated-RCE-FUXA-CVE-2023-33831
[3] https://www.exploit-db.com/exploits/49803
[4] https://pymodbus.readthedocs.io/en/latest/
[5] https://github.com/thiagoralves/OpenPLC_v3
[6] https://github.com/frangoteam/FUXA

The HMI, implemented with FUXA, is an interface for visualizing SCADA processes and serving as an interactive dashboard. Lastly, the PLC, implemented with OpenPLC, executes the automation algorithm and, using the Modbus/TCP communication protocol, interprets the requests given by the HMI and controls the plant.

The Architecture Coordinator module instantiates the entire system using a well-defined configuration file. Interprets the configuration, deploys Docker containers for each module, establishes connections, and generates the architecture description graphs $\mathcal{G}_N = \{\mathcal{N}_N, \mathcal{E}_N\}$, composed of every device in the architecture. Note that $\mathcal{N}_N$ is the set of network nodes n_N and $\mathcal{E}_N$ is the set of network connections $e_{N,N'}$ between nodes n_N and $n_{N'}$ where $N \neq N'$.

Moreover, the architecture also features the Data Collection module, which contains tools necessary to collect metrics, logs, and network data. Using proxies for each container, it monitors network activity and generates reports for normal operations and attack scenarios caused by the Attack Coordinator or other threats. This process produces essential datasets for IDS development.

Finally, the Attack Coordinator, the core focus of this work, plans and executes attacks on the simulated environment, and utilizes system instantiation data from the Architecture Coordinator. It connects to the Attack Database, which consists of an Attack Set comprising information detailed in Table 1 and an Attack Tool Set containing tools and exploit scripts for vulnerability exploitation.

3.1 Attack modeling and chains

The attack coordinator requires many data relations, from the attack requirements to the necessary precedence to compose the chains. Therefore, this section addresses how the modeling of both the attacks and the system will be handled.

Modeling the execution of attacks and attack chains relies on three crucial points: mapping vulnerability requirements, identifying the exploit techniques, and finding paths to turn the attack into a chain. The first two are mapped with examples presented in Table 1, which comprises information on targets, affected systems, and protocols, fulfilling the first requirement. It also contains references to exploits, which can vary from GitHub repositories, research papers, or white papers, meeting therefore the second requirement. However, due to precedence requirements of certain attacks, we consider the development of a custom ontology to be an effective approach for building these chains. An initial approach to the proposed problem is using Common Platform Enumeration (CPE), a naming scheme maintained by a reputable source (NIST) that uses XML and Uniform Resource Identifiers (URI). As systems become more complex and more specific contexts need to be identified, this can be augmented or even serve as groundwork for a custom ontology.

Considering the objective of producing a framework that executes attack chains, a graph shown in Figure 2 with the vulnerabilities from Table 1 was created as an example. This graph displays a possible attack chain considering an instantiated CPS and the studied attack vectors. Section 3.2 briefly describes the approach developed for modeling the attack chain graph (*i.e.*, finding an attack path within a CPS).

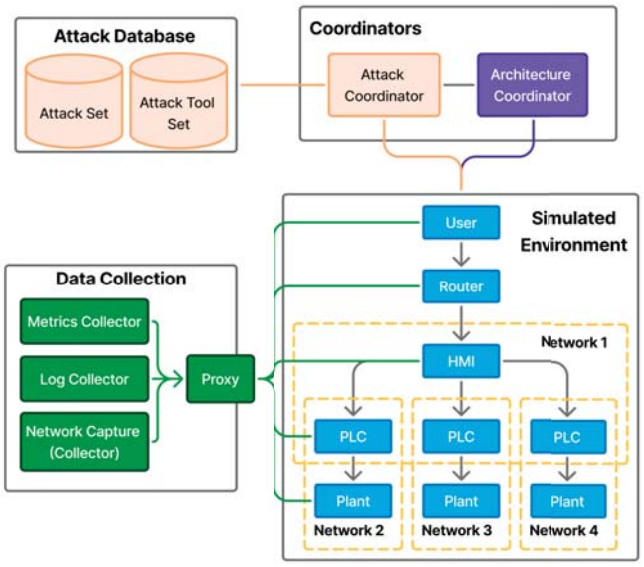


Figure 1: Architecture

3.2 Graph merger

The graph merger module of the attack coordinator combines the attack graph $\mathcal{G}_A = \{\mathcal{N}_A, \mathcal{E}_A\}$, composed by a set of available attacks, with the network graph $\mathcal{G}_N$ into a single graph which extrapolates any possible attack sequence to execute in the instantiated CPS. Note that $\mathcal{N}_A$ is the set of attacks (nodes) n_A of the chain and $\mathcal{E}_A$ is the set of attack connections $e_{A,A'}$ between n_A and $n_{A'}$ where $A \neq A'$.

The process consists of computing a comprehensive graph, known as merged graph, $\mathcal{G}_M = \{\mathcal{N}_M, \mathcal{E}_M\}$, that is obtained based on the strong product (from graph theory), which is the composition of the cartesian product and the tensor product. Note that $\mathcal{N}_M$ is the set of merged nodes n_{NA} that represent an attack n_A executed on a given network node n_N. $\mathcal{E}_M$, in turn, is the set of connections $e_{NA,N'A'}$ between nodes n_{NA} and $n_{N'A'}$ where $NA \neq N'A'$.

Algorithm 1 Graph Merger

1: **Input** : $\mathcal{G}_N, \mathcal{G}_A$
2: $\mathcal{G}_M \leftarrow \{\mathcal{N}_M, \mathcal{E}_M\}$
3: $Merger(\mathcal{G}_N, \mathcal{G}_A)$ {
4: $\mathcal{N}_M \cup \{n_{NA} \mid \forall n_N, n_A$ that meet network/attack requirements$\}$
5: $\mathcal{E}_M \cup \{e_{NA,N'A'} \mid n_{NA} \in N_M \wedge n_{N'A} \in N_M,\ \forall A, N \neq N'\}$
6: $\mathcal{E}_M \cup \{e_{NA,N'A'} \mid n_{NA} \in N_M \wedge n_{NA'} \in N_M,\ \forall N, A \neq A'\}$
7: $\mathcal{E}_M \cup \{e_{NA,N'A'} \mid n_{NA} \in N_M \wedge n_{N'A'} \in N_M,\ \forall (N \neq N' \wedge A \neq A')\}\}$
8: **Output** : $\mathcal{G}_M$

Algorithm 1 defines the general steps of the graph merger. It takes the foundations of the strong product and adapts it to be more efficient to our use case. Firstly, during the generation of the nodes of $\mathcal{N}_\mathcal{M}$, only those corresponding to n_A that are possible in n_N are added (e.g. an attack that targets a Modbus protocol wouldn't be possible in an HMI machine that does not support it therefore it wouldn't be added). This verification usually reduces a huge number of nodes, reducing therefore the number of edges that also need

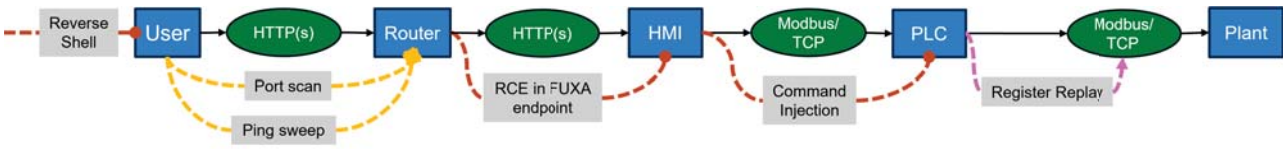


Figure 2: Attack chain example

to be verified in the next step. The next steps verifies the nodes that entered $\mathcal{G}_M$ and only adds in $\mathcal{E}_M$ edges that meet one of three requirements: has network or attack precedence, i.e. in the original graph it would have a connected edge, or if the combination has precedence. The first two requirements come from the cartesian product and the third from the tensor product. Finally, this algorithm outputs the final version of $\mathcal{G}_M$ from which the attack coordinator will randomly explore one of the chains to simulate a more or less experienced attacker, providing the necessary diversity inside the produced datasets. Later, an optimized attack path selection algorithm can be implemented. However this is not in the current scope of this work. Furthermore with the future use of an ontology elements mapped to certain categories would be grouped inherently improving even more the efficiency.

4 Conclusions and Forthcoming Work

This work aims to fulfill a significant gap in the current scope of CPS security: the lack of suitable datasets for ML-based IDS training. Continuous attack integration, followed by the production of CPS datasets with high variability in the system architecture and attack types, opens the possibility of implementing the new generation of IDSs for CPSs.

A set of singular attacks were tested, and their requirements are stored in a database. Three example end-to-end attack chains, i.e. from the entry point of the honeynet to the end, were formed and tested. Lastly, research and development of the defined algorithm for graph merger is ongoing. The next steps will be: structuring consistent ontologies to represent the attack and architectural characteristics needed for steps 3 and 4 of the graph merger algorithm; and implementing the tool for the automatic execution of attacks.

5 Acknowledgment

This work was supported by the 'Companhia Paranaense de Energia - COPEL' research and technological development program, through the PD-02866-0533/2021 project, regulated by ANEEL, and by FCT through the LASIGE Research Unit, ref. UID/408/2025.

References

[1] R. Baheti and H. Gill, "Cyber-physical systems," *The impact of control technology*, vol. 12, no. 1, pp. 161–166, 2011.

[2] A. Humayed, J. Lin, F. Li, and B. Luo, "Cyber-physical systems security—a survey," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1802–1831, 2017.

[3] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security & Privacy*, vol. 9, no. 3, pp. 49–51, 2011.

[4] E. F-Secure, "Blackenergy & quedagh, the convergence of crimeware and apt attacks," 2014.

[5] L. Spitzner, "Honeypots: catching the insider threat," in *19th Annual Computer Security Applications Conference*, pp. 170–179, 2003.

[6] I. Martins, J. Cecílio, P. M. Ferreira, and A. Oliveira, "Comparative analysis of cybersecurity datasets in industrial control systems," in *2024 IEEE MetroInd4.0 & IoT*, pp. 440–445, 2024.

[7] L. Sousa, J. Cecílio, P. M. Ferreira, and A. O. de Sá, "Reconfigurable and scalable honeynet for cyber-physical systems," *Ada Lett.*, vol. 44, p. 90–94, Dec. 2024.

[8] A. Rahman, G. Mustafa, A. Q. Khan, M. Abid, and M. H. Durad, "Launch of denial of service attacks on the modbus/tcp protocol and development of its protection mechanisms," *International Journal of Critical Infrastructure Protection*, vol. 39, p. 100568, 2022.

[9] T. Morris and W. Gao, "Industrial control system cyber attacks," in *Industrial Control System Cyber Attacks*, 09 2013.

[10] B. Chen, N. Pattanaik, A. Goulart, K. L. Butler-purry, and D. Kundur, "Implementing attacks for modbus/tcp protocol in a real-time cyber physical system test bed," in *IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR)*, pp. 1–6, 2015.

[11] S. Mauw and M. Oostdijk, "Foundations of attack trees," in *Information Security and Cryptology - ICISC 2005* (D. H. Won and S. Kim, eds.), pp. 186–198, Springer Berlin Heidelberg, 2006.

[12] S. Jha, O. Sheyner, and J. Wing, "Two formal analyses of attack graphs," in *15th IEEE Computer Security Foundations Workshop. CSFW-15*, pp. 49–63, 2002.

[13] X. Zou, "A survey on application of knowledge graph," *Journal of Physics: Conference Series*, vol. 1487, p. 012016, mar 2020.

[14] A. T. Al Ghazo, M. Ibrahim, H. Ren, and R. Kumar, "A2g2v: Automatic attack graph generation and visualization and its applications to computer and scada networks," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 10, pp. 3488–3498, 2020.

[15] Y. Qi, Z. Gu, A. Li, X. Zhang, M. Shafiq, Y. Mei, and K. Lin, "Cybersecurity knowledge graph enabled attack chain detection for cyber-physical systems," *Computers and Electrical Engineering*, vol. 108, p. 108660, 2023.

[16] Y. Jia, Y. Qi, H. Shang, R. Jiang, and A. Li, "A practical approach to constructing a knowledge graph for cybersecurity," *Engineering*, vol. 4, no. 1, pp. 53–60, 2018.

Task-to-Accelerator Mapping for Heterogeneous Systems Using Heuristics

Mohammad Samadi*, Tiago Carvalho, Luís Miguel Pinho
Polytechnic Institute of Porto, Portugal; email: {mmasa,tdc,lmp}@isep.ipp.pt
Sara Royuela
Barcelona Supercomputing Center, Barcelona, Spain; email: sara.royuela@bsc.es

Abstract

Heterogeneous hardware platforms can be used to improve the performance of computing systems in terms of application response time. They are in high demand, especially with the recent emergence of complex AI applications. Parallel applications can be used on these platforms to utilize most of the hardware system capacity and decrease the execution time. However, predictability is still a notable challenge on these platforms when used in time-critical systems due to the variability in the execution of parallel runtime systems, like OpenMP. Therefore, this paper proposes a new heterogeneous task-to-accelerator mapping approach using heuristics to improve the performance and predictability of OpenMP applications running on heterogeneous systems.

Keywords: parallel programming, OpenMP, heterogeneous systems, task-to-accelerator mapping, heuristics.

1 Introduction

Nowadays, graphics processing units (GPUs) are generally applied to accelerate diverse intensive computations, turning into general-purpose computing on GPUs (GPGPUs). In the Internet of Things (IoT) and Cyber-Physical Systems (CPS) domains, they can be applied to perform various tasks, such as pedestrian detection, eye tracking, and navigation [1, 2]. Besides, the performance of heterogeneous platforms can be improved using parallel programming models like OpenMP [3, 4, 5]. The timing and response time of programs running on the GPU should also be evaluated, similar to those running on the CPU. Therefore, an efficient task scheduling policy is necessary on these systems, especially for predictable OpenMP applications.

Contrary to traditional worst-case response time (WCRT) analyses for CPUs, analyzing heterogeneous systems is challenging because the number of processing elements (i.e., CPUs and GPUs) is high. Hence, mapping computation to processing elements becomes even more critical. Furthermore, replacement policies and exact cache architectures are not evident in modern GPUs as they offer a black-box model, as well as the effect of cache and memory is not considered while using long-delay operations (e.g., hundreds of cycles for an L1 cache miss) [6]. Consequently, it is essential to perform an efficient mapping process on the GPU side to overcome the problems that have arisen.

Predictability is one of the main requirements in time-critical systems (e.g., CPS). It can be achieved by reducing the variability of application response times. Accordingly, mapping tasks to processing elements (e.g., accelerator) in parallel applications (e.g., OpenMP) should be performed efficiently to improve the load-balancing of queues (in multi-queue systems like LLVM) and the work-conserving nature of the mapping. In this way, response time variability can be reduced and, consequently, system predictability can be improved.

This paper proposes an efficient task-to-accelerator mapping for multi-GPU systems considering OpenMP applications. In OpenMP, device tasks entail a CPU segment, which is scheduled based on the readiness of the task and the availability of host resources, and a GPU segment, which uses GPU resources to schedule and run the offloaded kernel. The proposed method focuses on the GPU segment, once these tasks are available to start executing on the GPU. The method first places the GPU task into a GPU global queue. Then, one of the local queues, individually dedicated to each GPU device, is selected for the task. Finally, the task is dispatched to the GPU to be executed. The selection policy of tasks from the global queue, the selection of the local queues, and the selection of tasks from local queues to run with the GPUs are determined using various heuristic algorithms. The main objective is to improve the predictability of real-time parallel and heterogeneous applications by reducing the response time variability (and to improve schedulability by minimizing WCRT) of the parallel runtime, focusing on the scheduling and mapping processes of the parallel runtime.

2 Related work

This section describes relevant methods presented when this work was submitted in the scope of task scheduling in heterogeneous systems.

Elliott et al. [7] presented GPUSync, a framework to improve parallel management of GPU-based computing resources. The framework manages the GPU budget and configures the system under different scheduling policies and prioritization schemes. Moreover, GPU resources are managed

*PhD Candidate, Universitat Politècnica de Catalunya, Barcelona, Spain

using job-level-static-priority (JLSP) schedulers, such as rate-monotonic (RM) and EDF. CPU and GPU allocations are performed under partitioned, clustered, and global basis models. This work does not enhance the load-balancing of FIFO queues, nor does it consider efficient algorithms to increase the degree of parallelism and minimize response time variability and WCRT.

Capodieci et al. [8] presented a prototype of a deadline-based scheduler with preemption policy for GPU tasks in a virtualized environment. This work uses a constant bandwidth server to provide task isolation in applications using the NVIDIA software stack. It undergoes a pre-existing table-based method for event-driven GPU command submission, uses real-time extensions for CUDA and OpenGL, and prototypes a software scheduler running as a privileged guest on the hypervisor. This approach, however, does not exploit execution-related information (e.g., task execution time) that could reduce response times efficiently.

Kim et al. [9] introduced a server-based approach to solve two problems related to GPU management, including busy waiting and long priority inversion. This approach considers multi-core platforms equipped with a single GPGPU through synchronous GPU access. GPU resources are shared between multiple tasks based on a non-preemptive sequential process, using fixed-priority partitioned scheduling and a single lock for accessing the GPU. Despite its benefits, this method considers a single GPGPU device, so tasks cannot be executed simultaneously using multiple accelerators to reduce application response time.

Tsog et al. [10] studied static allocation approaches through a fork-join model to reduce the bottleneck condition imposed by the overuse of GPUs. Accordingly, it performs offline static allocation of parallel computing segments on the CPU or GPU to evaluate the effect of balanced CPU and GPU utilization in real-time systems. In addition, the schedulability of tasks is improved using different heuristics, such as non-greedy resource allocation. Similar to the previous work, this method considers single-GPU systems. Furthermore, although techniques like support vector machine (SVM) are used for task scheduling, they are not considered in the evaluations because they entail a high overhead.

Olmedo et al. [11] introduced a hierarchical arbitration technique for GPU scheduling on NVIDIA platforms and considered the application scheduler at the top level of the scheduling hierarchy based on CUDA applications. Streams are pushed into a First In, First Out (FIFO) queue, as these blocks are distributed between the streaming multiprocessor (SM) cores using a block scheduler. Although this work has inferred various heuristics applied in the hierarchical scheduling scheme adopted with the NVIDIA GPU subsystem, it does not apply stream reordering techniques, limiting its benefits in GPU utilization and system predictability.

Amert et al. [12] presented the Time-Isolated Multicore Execution With AcceLerator Locking (TimeWall) framework to provide time partitioning on multi-core + accelerator platforms. This framework consists of three elements: a table-driven scheduler for allocating time to partitions, in-partition global EDF schedulers, and a special locking protocol for arranging the accelerator accesses based on the time-slice boundaries. This method expects the CPUs to wait until GPU operations in the accelerator are completed (synchronous offloading in OpenMP), which can considerably increase task execution time and application response time.

Overall, the main limitations exposed with the existing methods are that most of them (i) consider a suspension-oblivious analysis, where the CPU is not free while the offloaded part is executing in the GPU, so utilization loss and response time of the application may increase, (ii) focus on single-GPU systems, limiting their applicability in modern clusters and advanced (domain or zonal) E/E architectures for autonomous vehicles, and (iii) do not employ efficient advanced algorithms for selecting appropriate queues and tasks to optimize task scheduling based on its characteristics (e.g., execution time) and to reduce application response time. In addition, some do not consider multiple queues for GPU devices, which can increase contention on the single queue and reduce the system's predictability. Accordingly, this paper addresses problems by proposing a heterogeneous mapping through the multi-queue system model and a suspension-aware analysis for predictable parallel applications using several heuristics with different characteristics.

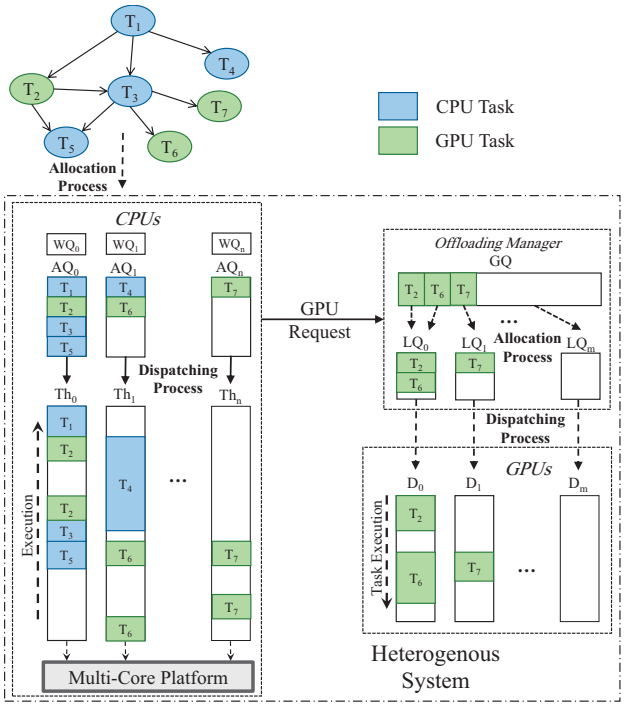
3 System model

This work assumes a multi-core platform with one or multiple GPU devices suitable for modern time-critical systems (e.g., CPSs). The system is supposed to have two types of tasks: CPU tasks and GPU tasks. CPU tasks are executed using only CPU cores, while GPU tasks are executed using both CPU and GPU (called host and device, respectively) resources. GPU tasks are dispatched to GPU(s) sequentially and non-preemptively. Furthermore, memory copy time values are task-specific.

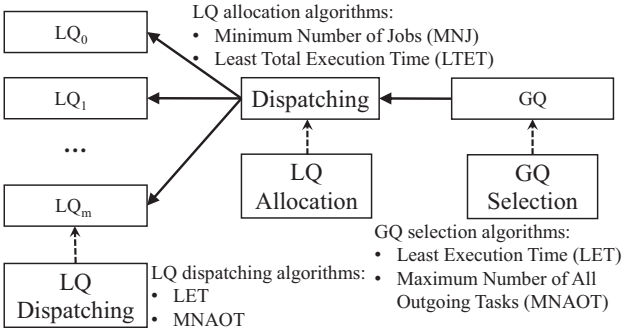
A task τ_i is characterized as $\tau_i = (T_i, D_i, C_i, G_i, \xi_i, \sigma_i)$, where T_i is the period of each job of the task, D_i is the relative deadline, C_i is the sum of the execution time of CPU execution segments, G_i is the execution time of GPU execution segment, ξ_i is the memory copy time from host to device, and σ_i is the memory copy time from device to host. Tasks are managed using a directed acyclic graph (DAG), where they may involve data dependencies on each other. In addition, the deadline is used in the system model to evaluate the schedulability of the system based on the obtained response time and the relative deadline.

4 Proposed approach

Figure 1 shows the proposed mapping methodology, including an overview of the system and the algorithms applied during the offloading process. As illustrated in Figure 1a, each CPU has an allocation queue *AQ* to execute CPU tasks (in blue) and the CPU execution segments of GPU tasks (in green). In addition, it has a waiting queue *WQ* to execute the second CPU segment of GPU tasks after the GPU segment is completed on the GPU. The task execution order in each GPU device is from top to bottom. For example, T_2 is executed before T_6 in the first device (i.e., D_0).



(a) System overview



(b) Offloading algorithms

Figure 1: Methodology of the proposed task-to-accelerator mapping.

The offloading process is carried out with an offloading manager that contains a global memory queue *GQ* and multiple local memory queues *LQs* dedicated to the GPUs. Since *LQ* includes a limited capacity, it is assumed that the *GQ* is shared between all *LQs*. GPU execution segments are first allocated to the *GQ* with CPUs (through the offloading manager) and then propagated (and allocated) to *LQs* using the offloading manager. Note that multiple tasks can be executed on different GPUs concurrently. The process is performed asynchronously, so a suspension-aware analysis is performed. Therefore, the CPU assigned to a GPU task is not suspended while the GPU segment runs, so that it can run other tasks. Furthermore, each GPU includes an independent lock controlled by the offloading manager, in which access to the GPU is locked while a kernel is running. Figure 1b shows three main processes performed using the offloading manager: GQ selection, allocation, and dispatching.

GQ selection: As mentioned above, each *LQ* has a limited capacity, where the capacity is the size of the queue in number of tasks, so GPU execution segments first are allocated to the *GQ* and then allocated to one of the *LQs*. When the *GQ* is not empty and at least one of the *LQs* has blank space to get a new task, a GPU segment is allocated to an *LQ*. Multiple tasks can exist in the *GQ* simultaneously, so one of the tasks is selected for each allocation using the GQ selection process. Accordingly, two algorithms are suggested for this process: Least Execution Time (LET) and Maximum Number of All Outgoing Tasks (MNAOT). LET selects the task with the shortest (GPU segment) execution time, and MNAOT selects the task with the most significant number of outgoing nodes in the Task Dependency Graph. They can improve the work-conserving nature of the mapping process by executing thin tasks and tasks with more successors first, respectively, and increasing the number of ready tasks in the system.

LQ allocation: If the number of *LQs* that include a blank space is more than one when allocating a task from the *GQ* to the *LQ*, one of the *LQs* is selected. LQ allocation selects the GPU to map the task given by the GQ selection process. For this purpose, two LQ allocation algorithms are suggested: Minimum Number of Jobs (MNJ) and Least Total Execution Time (LTET). MNJ selects the *LQ* with the fewest tasks, and LTET selects the *LQ* with the smallest total execution time [1]. These algorithms can improve the load-balancing of queues based on the number of tasks and execution time, respectively, to speed up the task execution process and reduce application response time and variability.

LQ dispatching: If a GPU device is idle and its *LQ* is not empty, one of the tasks in the *LQ*, if more than one, should be selected. Two LQ dispatching algorithms are suggested to select the most appropriate task among the existing ones. As the process of selecting a task is the same for the *LQs* and the *GQ*, the LQ dispatching algorithms are the same as those used in the GQ selection process. The mapping process on the CPU side is performed using heuristic-based task-to-thread mapping algorithms presented in a previous work [13].

Since two heuristics are used in each process, eight combinations of the algorithms can be considered for the mapping process. Therefore, the goal is to identify the best heuristics using evaluation results.

Task execution time can be determined using a randomized process in random DAGs and profiling tools (e.g., Extrae [14]) in benchmarks (i.e., synthetic and real-world applications). As the proposed heterogeneous mapping approach is presented for time-critical systems implemented with OpenMP, the worst-case execution time (WCET) is used to calculate task execution time based on multiple executions.

Furthermore, the application response time can be determined using two timestamps, one before and one after the main code block. After running the application several times, response times can be analyzed using a boxplot to determine response time variability and WCRT. In this way, the system's predictability and schedulability can be well studied.

[1]The total execution time of each *LQ* should be updated before this process.

5 Conclusions and future work

This paper introduces the preliminary design of a heterogeneous task-to-accelerator mapping method to reduce application response time and variability, the main requirements in real-time parallel applications. Using the OpenMP accelerator model, GPU tasks are divided into a CPU segment, which schedules the task into an offloading manager once it is ready (i.e., all its input dependencies, if any, are resolved), and a GPU segment, which includes the kernel offloaded and executed in the GPU. The device-side methodology consists of a global queue and several local queues, each dedicated to a GPU. An offloading manager controls the whole process, where GPU segments are allocated to the global queue, then allocated to local queues, and finally executed using GPUs. The methods of selecting the most appropriate tasks from the global and local queues and choosing the most suitable local queues are performed using heuristic algorithms.

The main objectives of this work are to improve queue load-balancing by selecting suitable queues for tasks and to improve the work-conserving nature of the mapping process by selecting appropriate tasks from the queues. Different heuristic algorithms are applied at each stage of the mapping process to investigate which ones can achieve these objectives. Subsequently, response time variability can be reduced to improve the predictability of parallel applications, which is a fundamental requirement in time-critical systems.

A future work will use the evaluation results of the proposed method to refine and extend the work depicted in this paper by suggesting enhancements and new algorithms for GPU scheduling. Furthermore, the performance of the work will be evaluated under multiple applications with different configurations on supercomputers like the MareNostrum 5 accelerated partition (ACC) [2].

6 Acknowledgment

The authors thank the anonymous reviewers for their helpful comments and feedback. This work was supported by the KDT Joint Undertaking (JU) program under grant agreement No 101140216 (project MATISSE), co-funded by the European Union's Horizon Europe research and innovation programme and the Portuguese Fundação para a Ciência e a Tecnologia (FCT), and the HiPART project, with reference PID2023-148117NA-I00, financed by MICIU/AEI /10.13039/501100011033 and FEDER, UE.

References

[1] Z. He, Y. Sun, B. Wang, S. Li, and B. Zhang, "Cpu-gpu heterogeneous computation offloading and resource allocation scheme for industrial internet of things," *IEEE Internet of Things Journal*, vol. 11, no. 6, pp. 11152–11164, 2023.

[2] N. R. Srivastava and V. Mittal, "Applications of gpus for signal processing algorithms: A case study on design choices for cyber-physical systems," in *Emergence of Cyber Physical System and IoT in Smart Automation and Robotics: Computer Engineering in Automation*, pp. 147–159, Springer, 2021.

[2] https://www.bsc.es/supportkc/docs/MareNostrum5/overview/

[3] D. Sevim, B. Bilgin, and I. Akturk, "Evaluating performance and energy efficiency of parallel programming models in heterogeneous computing systems," in *2024 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 309–319, IEEE, 2024.

[4] C. Yu, S. Royuela Alcázar, and E. Quiñones Moreno, "Enhancing heterogeneous computing through openmp and gpu graph," in *The 53rd International Conference on Parallel Processing (ICPP'24): August 12th-15th, 2024, Gotland, Sweden: main conference proceedings*, pp. 534–543, Association for Computing Machinery (ACM), 2024.

[5] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "Starpu: a unified platform for task scheduling on heterogeneous multicore architectures," in *Euro-Par 2009 Parallel Processing*, pp. 863–874, Springer, 2009.

[6] D. Ernst, G. Hager, M. Knorr, G. Wellein, and M. Holzer, "Opening the black box: Performance estimation during code generation for gpus," in *2021 IEEE 33rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 22–32, IEEE, 2021.

[7] G. A. Elliott, B. C. Ward, and J. H. Anderson, "Gpusync: A framework for real-time gpu management," in *2013 IEEE 34th Real-Time Systems Symposium*, pp. 33–44, IEEE, 2013.

[8] N. Capodieci, R. Cavicchioli, M. Bertogna, and A. Paramakuru, "Deadline-based scheduling for gpu with preemption support," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, pp. 119–130, IEEE, 2018.

[9] H. Kim, P. Patel, S. Wang, and R. R. Rajkumar, "A server-based approach for predictable gpu access with improved analysis," *Journal of Systems Architecture*, vol. 88, pp. 97–109, 2018.

[10] N. Tsog, M. Becker, F. Bruhn, M. Behnam, and M. Sjödin, "Static allocation of parallel tasks to improve schedulability in cpu-gpu heterogeneous real-time systems," in *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1, pp. 4516–4522, IEEE, 2019.

[11] I. S. Olmedo, N. Capodieci, J. L. Martinez, A. Marongiu, and M. Bertogna, "Dissecting the cuda scheduling hierarchy: a performance and predictability perspective," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 213–225, IEEE, 2020.

[12] T. Amert, Z. Tong, S. Voronov, J. Bakita, F. D. Smith, and J. H. Anderson, "Timewall: Enabling time partitioning for real-time multicore+ accelerator platforms," in *2021 IEEE Real-Time Systems Symposium (RTSS)*, pp. 455–468, IEEE, 2021.

[13] M. Samadi, S. Royuela, L. M. Pinho, T. Carvalho, and E. Quiñones, "Time-predictable task-to-thread mapping in multi-core processors," *Journal of Systems Architecture*, vol. 148, p. 103068, 2024.

[14] "Extrae, bsc performance tools," 2024.

Reasoning about Subprogram Termination in SPARK

Claire Dross, Joffrey Huguet, Johannes Kanig
AdaCore, 75009 Paris, France; email: {dross, huguet, kanig}@adacore.com

1 Introduction

The Ada language is designed for safety critical applications. However, most of its safety-oriented features – language mandated checks and contract based programming in particular – rely on dynamic checks. The SPARK tool performs formal proof of Ada programs, verifying their correctness for all possible inputs. It verifies statically both the language mandated checks and conformance to user-provided contracts. The analysis performed by SPARK is modular, operating on a per-subprogram basis. To model interactions between subprograms, it relies on contracts, making it necessary for users to annotate their subprograms carefully. Preconditions define the valid calling contexts, they are assumed when verifying the subprogram, and checked on calls. Conversely, postconditions summarize the effect of a subprogram call when it returns normally. They are checked when the subprogram is verified, and assumed when verifying callers. As the analysis is modular, only the information that is explicit in the postcondition of a subprogram will be known when verifying callers [1].

However, these classical contracts might not be enough to describe the behavior of an Ada subprogram. Indeed, calls in Ada can exhibit various behaviors beyond a normal return. They might loop forever, propagate exceptions, or even cause the entire program to terminate abruptly. Because the analysis of SPARK is modular, such behaviors need to be described in the subprogram's signature, so it can be used when verifying callers. Additional subprogram contracts have been designed to address these different scenarios. The following sections provide an overview of these contracts and explain how they can be used to represent the various ways in which a subprogram might conclude its execution.

2 Annotating and Proving Termination

In general, the SPARK tool enforces partial correctness, that is, it verifies the program but does not enforce termination. However, termination might be a property of interest for programmers. To instruct the tool to verify that a subprogram terminates, users can annotate it with the `Always_Terminates` aspect, as demonstrated in Fig. 1. It is also possible to put the annotation on a package, in which case it applies to all subprograms declared in the package.

When a subprogram with the `Always_Terminates` aspect is analyzed, the SPARK tool looks for potentially non-terminating constructs in its body. This includes subprogram calls, as well as loops. To make the feature easier to use, the tool infers termination annotations when they are not supplied on called subprograms. For example, in Figure 1, it is not mandatory to annotate `P_Called` with an `Always_Terminates` aspect for the verification of `P` to succeed. The SPARK tool implicitly attempts to prove that it terminates, and reports an error on the call in `P` if the verification fails.

```
procedure P_Called;
procedure P with Always_Terminates is
begin
   P_Called;
end P;
```

Figure 1: A subprogram annotated with `Always_Terminates`

```
function Compute_Log (C, N : Positive) return Positive with
  Pre => C > 1 and N <= Positive'Last / C
is
  Y : Positive := C;
  Res : Positive := 1;
begin
  while Y * C <= N loop
    Y := Y * C;
    Res := Res + 1;
    pragma Loop_Variant (Increases => Y);
  end loop;
  return Res;
end Compute_Log;
```

Figure 2: A loop annotated with a variant

Concerning loops that might not terminate in Ada (*for* loops on arrays and ranges necessarily terminate in Ada), it is necessary to provide additional guidance for the termination annotation to be verified, in the form of a *variant*. In most verification-aware languages, variants are non-negative integer values that strictly decrease at each iteration. Since a non-negative integer value might only decrease a finite number of times, this is enough to ensure that the loop terminates. For SPARK, variants might also be signed integers of a fixed size that either strictly increase or strictly decrease at each iteration. It is enough, as SPARK verifies that overflows cannot occur in computations. As an example, the SPARK tool can verify that the loop in the `Compute_Log` function in Figure 2 terminates, as *Y* is a signed integer that strictly increases at each iteration of the loop.

```
procedure P1 (X : Integer) with
  Subprogram_Variant => (Decreases => X, Increases => 1)
is
begin
  P2 (X);
end P1;

procedure P2 (X : Integer) with
  Subprogram_Variant => (Decreases => X, Increases => 2)
is
begin
  if X > 0 then
    P1 (X – 1);
  end if;
end P2;
```

Figure 3: Recursive subprograms annotated with subprogram variants

In addition, for pointer-based recursive data structures, *structural variants* are possible. Since such data-structures are necessarily acyclic in programs verified by SPARK, it is enough for the tool to check that recursive calls necessarily occur on strict subcomponents of the variant expression.

Verifying termination of recursive calls also requires providing variants. When the tool detects that a call might introduce recursion, it requires that the user provides a matching variant on all mutually recursive subprograms. Checking that the variant *progresses* (decreases or increases) for each mutually recursive call is enough to ensure termination. Note that in general, it would only be necessary for a variant to progress once in every cycle of the chain of recursive calls, but it is notably harder to check modularly. As an example, the procedure `P1` in Figure 3 calls `P2` which calls `P1` again. To ensure termination, it is not theoretically necessary for a variant to progress both when `P1` calls `P2` and when `P2` calls `P1`, but the SPARK tool mandates it so verification can be done modularly.

To workaround this restriction, it is possible for the user to introduce static variants, in addition to the normal ones. Indeed, SPARK supports providing a sequence of variants that should progress lexicographically - either the first one progresses, or it stays the same and the second one progresses. By adding a static integer value as a second variant, it is possible to dispense some calls in the recursive chain from making the first variant progress. As an example, in Figure 3, it is possible to call `P2` directly on `X` inside `P1`, as the second variant increases, but `P2` might only call `P1` on a strictly smaller value of `X`.

The `Always_Terminates` aspect is associated with a boolean expression. When none is supplied, the default value is True, which means that the subprogram shall terminate on all inputs. Using a value of False is in general useless, as not enforcing termination on any input is the default for SPARK. More generally, it is possible to use a dynamic boolean value in an `Always_Terminates` aspect. This expression is evaluated at the beginning of the subprogram (it is a precondition), so it can refer to inputs of the subprogram but not to outputs. The subprogram shall terminate whenever the condition evaluates to true. Otherwise, it does not need to, but it might. The

```
procedure Necessarily_Terminates with
  Always_Terminates;

procedure Might_Not_Terminate with
  Always_Terminates => False; -- This is the default

procedure Connect_With_Retry (Nb_Try : Natural := 0) with
  Always_Terminates => Nb_Try /= 0;
```

Figure 4: A non-trivial `Always_Terminates` condition

```
procedure P (C : Color) with Pre => C /= Blue is
begin
  case C is
    when Red =>
      ...
    when Blue => raise Program_Error;
  end case;
end P;
```

Figure 5: Defensive coding using exceptions

procedure `Connect_With_Retry` in Figure 4 for example takes as a parameter the number of times it is allowed to try connecting to a socket in case of failure. It will necessarily terminate unless its parameter is given the special value 0, which stands for indefinitely.

3 Raising, Handling, and Propagating exceptions

Exceptions in Ada are used for different purposes. First, an exception will be raised implicitly whenever a language mandated check (an overflow, an index check...) or a user-supplied contract (pre or postcondition...) fails at runtime. This is not a case that we consider in SPARK, as the tool proves statically language mandated checks and user-supplied contracts - a different choice is sometimes made in other verification tools, as explained in Section 2. Therefore, we concentrate on explicitly raised exceptions in the rest of the section.

In Ada, it is common to raise exceptions in defensive coding, to signal program points that should be unreachable during the normal execution of the program. In this case, the SPARK tool should attempt to prove that the raise statement never occurs instead of taking into account the exceptional control flow. Figure 5 gives an example of a classical use case for exceptions as defensive coding. The exception `Program_Error` is raised to signal an unexpected case in a conditional. Exceptions might also be used to signal that a particular event occurred during the evaluation of a subprogram to its callers. It is the case for example of subprograms from the `Ada.Text_IO` standard library, like the `Open` function that might propagate `Name_Error` if no external file with the given name exists.

To handle these two use cases correctly, SPARK makes a difference between *expected* and *unexpected* exceptions. If an exception is unexpected, the tool tries to prove that the raise statement is unreachable, while if it is expected, it accounts for the exceptional control flow. An exception is considered to

```
procedure Open (File : in out File_Type;  ...)  with
  Exceptional_Cases => (Name_Error | Use_Error => True),
  Pre => not Is_Open (File);
```

Figure 6: Contract about expected exceptions in the standard library

```
procedure Read_Value (F : in out File_Type; V : out Value) with
  Pre => Is_Open (F),
  Exceptional_Cases => (others => not Is_Open (F));
```

Figure 7: Subprogram annotated with an exceptional postcondition

be expected if it is either handled locally in the subprogram, or if it is among the exceptions that the subprogram is supposed to propagate. For the tool to be able to make the distinction, it is necessary for the user to annotate subprograms with an `Exceptional_Cases` aspect that lists all the exceptions that might be propagated by the subprogram. As an example, Figure 6 shows the contracts that can be supplied for `Open`. Note that this contract does not list the `Status_Error` as an expected exception. Indeed, a precondition prevents `Open` from being called on files that are already open, so raising `Status_Error` in `Open` should be considered defensive coding. By default, subprograms are not expected to propagate any exceptions in SPARK. Therefore, it is not necessary to supply an exceptional contract for the procedure `P` from Figure 5 for the SPARK tool to consider that `Program_Error` is unexpected and check that it is not raised.

It is possible to write imprecise `Exceptional_Cases` contracts using the `others` keyword. This means that the subprogram is allowed to propagate any exception. This feature can be used to avoid the blow-up in exception annotations that could occur if all subprograms were allowed to propagate to their callers the exceptions raised by the subprograms they call.

As the SPARK toolset works modularly on a per subprogram basis, specific postconditions might be necessary to describe the effect of a subprogram call when it propagates an exception, so as to be able to verify code that handles it. For this purpose, it is possible to associate an exceptional postcondition with each expected exception in the `Exceptional_Cases` aspect. As an example, the procedure `Read_Value` in Figure 7 attempts to read a value from a file. Its `Exceptional_Cases` aspect states that it might propagate some exceptions that are left unspecified, using the `others` keyword. The associated boolean expression makes it possible to verify potential exception handlers that might surround calls to `Read_Value`. It states that, when an exception is propagated, the file has been closed by the procedure. Because they might not be copied back on exceptional exit, exceptional postconditions cannot mention the final value of parameters of the subprogram unless they are known to be passed by reference.

Currently, the `Exceptional_Cases` aspect does not have runtime semantics. The compiler could be extended to generate code to check exceptional postconditions at runtime when exceptions are enabled.

```
procedure OS_Exit (Status : Integer) with
  No_Return, Program_Exit, Always_Terminates;
-- Exit to OS with given status code (program is terminated)
```

Figure 8: Fully annotated `OS_Exit`

4 Abruptly Terminating the Program

It is also possible for a subprogram to cause the whole program to terminate abruptly with a status code. This is not common in Ada, but it might happen in particular when the Ada program calls subprograms written in another language, like C for example. Remark that raising an exception is never considered to abruptly terminate the program by the SPARK tool. As described in the previous section, exceptions are always expected to be potentially caught in callers. Treating exceptions specifically when handlers are not allowed, or on runtimes where exceptions cannot be handled, is a potential future improvement.

Like for exceptions, subprograms that might abruptly exit the whole program shall be annotated when using the SPARK toolset. The `Program_Exit` aspect indicates that a subprogram might exit the program. If a subprogram does not have this annotation, the tool verifies that it cannot cause the program to terminate abruptly, so abrupt program exit can also be used in defensive coding. As an example, Figure 8 gives the specification which could be used for the procedure `OS_Exit` of the `OS_Lib` library of GNAT that terminates the program with the status code passed as parameter. To state that the procedure necessarily exits the whole program, the aspect `Program_Exit` is not enough. It expresses that the subprogram might exit the program, but not that it necessarily does. The aspects `No_Return` and `Always_Terminates` prevent it from respectively returning normally and looping forever.

Like for `Exceptional_Cases`, it is possible to associate a boolean expression with a `Program_Exit` aspect. It is used as a postcondition for cases where the program is terminated: if there is a boolean expression, SPARK verifies that it holds when the program is terminated abruptly. These postconditions are useful to express properties that a user might want to enforce when the program is terminated, to register the exit code for example. As an example, Figure 9 uses a ghost function - a function that is removed when assertions are disabled - to represent the error status in case of program exit. Since nothing is executed after a program exit, subprogram outputs can be left in an invalid state. Only those outputs that are mentioned in the program exit postcondition - `Error_State` here, need to be valid.

5 Specifying Termination Based on Inputs

As explained in previous sections, using additional aspects, it is possible to associate a postcondition to each path exiting a subprogram - be it normally, with an exception, or by terminating the program. Even if these contracts are effectively postconditions and not preconditions - they are evaluated on

```
function Error_Status return Integer with Ghost,
  Global => Error_State;  --
Ghost state to model termination info

procedure OS_Exit (Status : Integer) with
   Global => (In_Out => Error_State),
   No_Return,
   Always_Terminates,
   Program_Exit => Error_Status = Status;
```

Figure 9: Subprogram annotated with a postcondition on program exit

```
procedure Raise_E_If_Cond (Cond : Boolean) with
  Post => not Cond,
  Exceptional_Cases => (E => Cond);

procedure Add_Digit_To_Res (C : Character) with
  Post => C in '0' .. '9'
  and then Res'Old * 10 + To_Int (C) in Natural'Range
  and then Res = Res'Old * 10 + To_Int (C),
  Exceptional_Cases =>
    ( Invalid_Digit  =>
       C not in '0' .. '9' and then Res = Res'Old,
     Overflow       =>
       C in '0' .. '9'
         and then
       Res'Old * 10 + To_Int (C) not in Natural'Range
         and then Res = Res'Old);
```

Figure 10: A combination of postcondition and exceptional postconditions

subprogram exit - they make it possible to describe how a subprogram is allowed to terminate depending on its inputs. This is achieved by introducing conjuncts that only constrain the subprogram inputs: for such a condition to hold on subprogram exit, it needs to also hold on entry. It means that the subprogram cannot exit in the associated way if the condition evaluates to False on subprogram entry. As an example, consider the subprogram `Raise_E_If_Cond` in Figure 10. Together, its normal and its exceptional postconditions enforce that it necessarily raises the exception `E` if its parameter `Cond` is true and returns normally otherwise.

Even if they are expressive enough to do so, using postconditions to express these guards makes them harder to read. They often result in the information being spread-out and potentially duplicated between the various kinds of postconditions. In addition, these guards might end up being merged with actual postconditions describing the effect of the subprogram on outputs. This is demonstrated by the contract of `Add_Digit_To_Res` in Figure 10. The procedure reads a digit as a character, and adds it to the number `Res`, formed of previously read digits. If an exception is propagated, `Res` is left unchanged. This is expressed in the third line of the postcondition for normal behavior, and at the end of each case in the `Exceptional_Cases` aspect. The rest of the contract specifies cases that lead to exceptions being propagated: if the character is not a digit, or if the computation overflows. The fact that this information is separated between the normal and exceptional postconditions makes it harder to read, and actual postconditions are buried under this information.

```
procedure Add_Digit_To_Res (C : Character) with
  Post => Res = Res'Old * 10 + To_Int (C),
  Exceptional_Cases => (others => Res = Res'Old),
  Exit_Cases =>
    ((C not in '0' .. '9')
       => (Expection_Raised => Invalid_Digit),
    (C in '0' .. '9')
     and then (Res * 10 + To_Int (C) not in Natural'Range)
       => (Exception_Raised => Overflow),
     others => Normal_Return);
```

Figure 11: A simplified version of the contracts using `Exit_Cases`

To improve readability for this common pattern, the `Exit_Cases` aspect has been introduced. It allows describing directly the expected behavior of a subprogram based on its inputs, by providing pairs of a guard and an *exit kind* - normal return, propagated exception, or abrupt program exit. The SPARK tool ensures that, when the subprogram exits, there is at most one guard that evaluates to True, and, if there is such a guard, that the subprogram necessarily exits in the way associated with it. Figure 11 shows how it can be used to simplify the contract of `Add_Digit_To_Res`. Both its normal and exceptional postconditions are now easier to read, as they only give information about subprogram outputs. The `Exit_Cases` aspect describes how the procedure terminates based on its inputs. The use of the Old attribute to refer to the value of `Res` at the beginning of the call is no longer necessary as guards are handled as preconditions. The use of the `others` case avoids duplication.

Note that the `Exit_Cases` aspect is only syntactic sugar for a combination of normal, exceptional, and program exit postconditions. It can be coupled with an `Always_Terminates` aspect to ensure that the subprogram actually terminates.

6 Related Work

As exceptions are ubiquitous in Java, verification tools for this language, such as OpenJML [2, 3] or KeY [4], generally support exceptions in a more general way than what we are doing here. In particular, these tools allow reasoning about the raised exception as a value, allowing subtyping for example, to state that a subprogram might propagate any exception of a given class. This is not something allowed by the Ada language, as exceptions are not regular values. Both tools have made different choices concerning implicit exceptions raised in case of failed language mandated checks. While OpenJML, like SPARK, attempts to verify that these checks never fail, KeY on the other hand allows propagating and handling such exceptions [5]. It has the advantage of allowing the verification of more subprograms, but might also result in defects being unintentionally hidden by an enclosing handler. In Ada, where exceptions are used less often, it was decided to favor simplicity over expressivity in this case.

More generally, other verification tools use contracts similar to those we introduced in SPARK to reason about nontermination and abnormal termination. The Java Modeling Language (JML), used by formal verification tools for Java, provides syntax for normal postconditions, through *ensures*

clauses, and exceptional postconditions, expressed by *signals* clauses [6]. The ACSL annotation language for C [7], used by the Frama-C tool [8], introduces *exit* clauses that provide properties that should hold when the function terminates the program. There is also a special construct to refer to the exit status in these contracts, similar to the ghost function introduced for the postcondition of `OS_Exit` in Section 4.

Finally, while most verification tools introduce postconditions for abnormal termination, shorthands that can be used to express how a subprogram might terminate based on its inputs like `Exit_Cases` are less common. The Byteback tool for formal verification at the level of Java bytecode has chosen to introduce shorthands to specify when a method returns normally or propagates an exception [9]. Generally, it offers a single postcondition that can be used to specify properties that need to hold both on the normal and the exceptional exit. It takes as parameter the (potentially *Void*) propagated exception. To ease specification, two shorthands are introduced, *Raise* and *Return*. They allow users to state that a method needs to terminate or propagate an exception when a predicate on the method's inputs evaluates to True. This is rather close to what we are doing, even if the specification is split into several annotations, as opposed to a single case.

7 Conclusion

Historically, the SPARK tool has been primarily concerned with proving subprograms which terminate normally. Thanks to a number of additional aspects for contracts, it is now also possible to verify subprograms that loop forever, propagate exceptions or exit the whole program.

References

[1] C. Dross, "The work of proof in SPARK," *Ada User Journal*, vol. 43, no. 3, 2022.

[2] D. R. Cok, "OpenJML: JML for Java 7 by extending OpenJDK," in *NASA Formal Methods Symposium*, pp. 472–479, Springer, 2011.

[3] P. Chalin, J. R. Kiniry, G. T. Leavens, and E. Poll, "Beyond assertions: Advanced specification and verification with JML and ESC/Java2," in *Formal Methods for Components and Objects* (F. S. de Boer, M. M. Bonsangue, S. Graf, and W.-P. de Roever, eds.), (Berlin, Heidelberg), pp. 342–363, Springer Berlin Heidelberg, 2006.

[4] R. Abbasi, J. Schiffl, E. Darulova, M. Ulbrich, and W. Ahrendt, "Deductive verification of floating-point Java programs in KeY.," in *TACAS (2)*, pp. 242–261, 2021.

[5] J. Boerman, M. Huisman, and S. Joosten, "Reasoning about JML: Differences between KeY and OpenJML," in *14th International Conference, IFM 2018, Maynooth, Ireland, September 5-7, 2018, Proceedings*, pp. 30–46, 08 2018.

[6] P. Chalin, J. R. Kiniry, G. T. Leavens, and E. Poll, "Beyond assertions: Advanced specification and verification with JML and ESC/Java2," in *Formal Methods for Components and Objects* (F. S. de Boer, M. M. Bonsangue, S. Graf, and W.-P. de Roever, eds.), (Berlin, Heidelberg), pp. 342–363, Springer Berlin Heidelberg, 2006.

[7] P. Baudin, J.-C. Filliâtre, C. Marché, B. Monate, Y. Moy, and V. Prevosto, *ACSL: ANSI/ISO C Specification Language*.

[8] P. Baudin, F. Bobot, D. Bühler, L. Correnson, F. Kirchner, N. Kosmatov, A. Maroneze, V. Perrelle, V. Prevosto, J. Signoles, *et al.*, "The dogged pursuit of bug-free C programs: the Frama-C software analysis platform," *Communications of the ACM*, vol. 64, no. 8, pp. 56–68, 2021.

[9] M. Paganoni and C. A. Furia, "Reasoning about exceptional behavior at the level of Java Bytecode," in *Integrated Formal Methods* (P. Herber and A. Wijs, eds.), (Cham), pp. 113–133, Springer Nature Switzerland, 2024.

12th Ada Developer Room at FOSDEM 2025

Dirk Craeynest

KU Leuven, Dept. of Computer Science, B-3001 Leuven, Belgium; email: Dirk.Craeynest@cs.kuleuven.be

Fernando Oleo Blanco

Open Source & Ada aficionado; tel: +34 689 44 27 45; email: irvise@irvise.xyz

Abstract

FOSDEM is a huge open source event organized each year in Brussels, Belgium. Among others, it features dozens of tracks on specific topics: the s.c. Developer Rooms. This year, for the 12th time, there was a track about the Ada programming language and related technologies. A brief overview of the half-day "Ada DevRoom" program is given, followed by reports of all technical presentations by their respective authors.

Keywords: Free and Open Source, FOSDEM, Ada.

1 Introduction

FOSDEM [1], the Free and Open source Software Developers' European Meeting, is an annual non-commercial two-day weekend event organized early February in Brussels. It is highly developer-oriented and brings together 8000+ participants from all over the world. It is free to attend and no registration is necessary. The 25th edition on Saturday 1 and Sunday 2 February 2025 offered over 1000 lectures: keynotes, lightning talks, parallel main and BoF tracks, plus a stunning 70 Developer Rooms!

In the period 2006-2020, Ada-Belgium already organized 10 "in-person" Ada DevRooms at FOSDEM. In 2022, due to the COVID-19 pandemic, we had a full-day online Ada DevRoom. In recent years, competition between DevRoom proposals increased tremendously, so after a hiatus of 2 years, we were happy to obtain a half-day slot on Sunday morning 2 February 2025. Despite less available time, the Ada FOSDEM community organized once more a large variety of presentations on the Ada programming language, tools and projects: a total of 11 Ada-related presentations by 11 authors from 7 countries!

The event was organized in cooperation with Ada-Belgium and Ada-Europe, and saw a good level of participation and great discussions at and after the DevRoom. Ada-Belgium handed out free "Ada coin" stickers to all participants.

2 Program overview

The Ada DevRoom featured the following program:

- *Welcome to the Ada DevRoom*, by Fernando Oleo Blanco, Spain, and Dirk Craeynest, Belgium;
- *Updates on the Ada Ecosystem*, by Fernando Oleo Blanco, Spain;
- *Get started with Ada in 2 minutes or less!*, by A.J. Ianozi, USA;
- *Advent of Compression: writing a working BZip2 encoder in Ada from scratch in a few days*, by Gautier de Montmollin, Switzerland;
- *Ada and Mini-Ada: a solution to the two-language problem*, by Gautier de Montmollin, Switzerland;
- *Understanding liquid types, contracts and formal verification with Ada/SPARK*, by Fernando Oleo Blanco, Spain;
- *The state of Rust trying to catch up with Ada*, by Oli Scherer, Germany;
- *Cryptography in SPARK: building the foundation with constant-time bigints*, by César Sagaert and Fabien Chouteau, France;
- *Multiword Arithmetic and Parallel Computing*, by Jan Verschelde, USA;
- *Developing device drivers for Ironclad using Ada*, by Cristian Simon Felipe, Spain;
- *AdaBots - programmable minetest bots*, by Tama McGlinn and Rudolf Batke, the Netherlands.

The Ada at FOSDEM 2025 web page has all details, see:

https://fosdem.org/2025/schedule/track/ada/

It provides the full schedule, abstracts of presentations, biographies of speakers, and pointers to more info. Recordings of each presentation and Q&A, plus copies of the slides, are available there as well.

3 Informal proceedings in the AUJ

Authors were invited to prepare short papers based on their presentations, to be included in the Ada User Journal. We're glad they all accepted! The goal was to make it easy for them and to be as inclusive as possible, hence the Journal accepted papers of variable length, which for this edition ranged from 1 to 3 pages.

The collection of papers is provided in this AUJ issue, as informal FOSDEM 2025 Ada DevRoom proceedings.

[1] https://fosdem.org/2025/

Updates on the Ada Ecosystem from the Past Few Years

Fernando Oleo Blanco
Open Source & Ada aficionado; Tel: +34 689 44 27 45; email: irvise@irvise.xyz

Abstract

As it had been a few years since the Ada community had a Devroom in FOSDEM, the author saw fit to have a small overview of the major releases and changes that had taken place in the Ada ecosystem for the past few years. In this article, the main changes in Ada 2022, release of GNAT-LLVM, WASM usage, Alire 2.0, the Ada-Lang website, the creation of the Ada User Society and the information released by NVIDIA and the NSA regarding the usage of Ada. Finally, there is a small discussion regarding future changes and challenges.

The following article will summarize the main ideas and concepts presented in the FOSDEM 2025 talk of the same name as the article.

Keywords: SPARK, contracts, types, liquid types, data modeling

1 Ada 2022

The release of the Ada 2022 standard bring new changes to the languages, making it slightly more functional, ergonomic and bringing new tools to the table. The main features and changes are:

- We now have iterator filters using the **when** keyword, making loops much more robust and showing the intent more clearly.

```
for E of Some_Array when E /= 0 loop
    Put_Line (E'Image);
end loop;
```

- The 'Reduce attribute has been added. This adds some functional goodness to the language and makes working with arrays an easier task.

```
function Factorial (N : Natural) return Natural is
([ for J in 1..N => J ]'Reduce("*", 1));
```

- Declare expressions can be used in expressions, making some assignments and expression functions nicer to work with and more powerful.

```
Val : Integer := (declare X : constant Integer := Random_Val;
                  begin X + X);
```

- The Static attribute can be used to mark expression functions that can be evaluated at compile time. These functions can be used anywhere in the code where static values are needed. This can also be used to precompute values during compilation, potentially reducing the code of the generated binary and making it run faster.

```
function Half_Size (S : Integer) return Integer is (S / 2)
   with Static;
```

- Compiler generated 'Image for all datatypes, which allows us to present the data in ASCII easily.

```
-- It does not need to be a simple numeric value
My_Subtype'Image (My_Object);
-- Put_Image attribute for user-defined printing
My_Special_Type is ...
   with Put_Image => My_Custom_Printing_Function;
```

- Delta aggregates, which allows to update and/or copy fields of a **record** much more easily.

```
Record_Object := (Record_Object with delta
                  Field1 => X, Field2 => Y);
```

- We can now use array indexes to create inline construction of array aggregates.

```
Squares_Array : Array_Type :=
   [for I in Index_Type => I * I];
```

- We can now refer to the target name using @, which prevents repeating the target name and creating long expressions.

```
Some_Very_Long.And_Complex (Expression) := @ + 1;
```

- The addition of the parallel construct and keyword. This new construct indicates to the compiler that the iteration (could be **for** or 'Reduce) can take place in parallel.

```
parallel
for I in Grid'Range(1) loop
    Grid(I, 1) := (for all J in Grid'Range(2) => Grid(I,J) = True);
end loop;
```

- The addition of Big_Numbers package to Ada.Numerics. There are Integers and Real child packages.
- Additional packages for atomic operations.
- A simple support for variadic C functions.
- The addition of the Jorvik profile, which loosens up some restrictions of the Ravenscar profile.
- The addition of the Nonblocking aspect to indicate that the function or procedure must not contain or call to potentially blocking code.

2 GNAT-LLVM

There has been a new compiler in town for a while. Ada code can now be compiled with the LLVM backend [1]. This new compiler is called GNAT-LLVM and it is using the GNAT frontend and generating LLVM IR, which can then be compiled to anything that LLVM supports. This brings a large ecosystem to the Ada language. It is known that this new compiler has been used to compile Ada for NVIDIA GPUs (using CUDA as an intermediary) and for WASM [2]!

The latest release of GNAT-LLVM, at the time of writing, supports up-to LLVM-19.

3 Web-Assembly

Web-Assembly, or more commonly known as WASM, started with the goal of improving computational speed in the browser. It was meant to be a tool to speed up JavaScript by creating a generic, low-level assembly that could run on a sandbox and communicate with JS to transfer inputs and outputs. However, it has been growing bigger and more powerful. There are now several implementations of WASM engines within and outside of web-browsers; it is also being eyed for fast operations within sandboxed, fast start-up VMs in cloud environments and for IoT: and it is a fully formally defined and provable language. The latest standard of WASM, at the time of writing is WASM 3.0.

With the new GNAT-LLVM compiler, it is now possible to compile Ada to WASM using the AdaWebPack [3] runtime. AdaWebPack is based on `bb-runtimes` and while it is a bit limited for the time being, it can already do quite a lot!

An impressive application already making use of this technology is G-NAV, a glider guiding and navigation system created by Guillermo Hazebrouk [4].

4 SPARK

SPARK [5] itself has also been slowly but steadily improving, to the point where it is almost no longer an Ada subset. The new improvements include:

- The addition of exception analysis, exception propagation and handling with the new Exceptional_Cases aspect.
- Improved its termination analysis capabilities.
- Upgraded to the CVC5 prover and updated the Z3 and alt-ergo provers.

It is hoped that in the near future it will gain the ability to do light-weight controlled types with the newly added, GNAT-specific, Finalizable aspect.

Also, a reminder, there is the `Apache v2` license SPARKlib [6] toolsuite, which brings a wide-range of SPARK-proven data structures and algorithms!

5 NVIDIA

Wow, just wow, NVIDIA has been on a roll in the markets because of the AI boom and... Oh, wait, no no, this is an Ada centered journal, then... Why is NVIDIA mentioned here? Because of Ada! They are probably nowadays the largest users and creators of Ada code in the world. During FOSDEM I showed the announcements of NVIDIA making use of Ada which had been released by that date. However, just a few months later we got much bigger news.

NVIDIA has certified to SIL-4 a seven (7) million lines Operating System, written in Ada/SPARK. This is done in the ISO-26262 automotive framework for self-driving cars, source: `https://www.eenewseurope.com/en/nvidia-drives-ada-and-spark-into-driverless-cars/` Additionally, NVIDIA published their report/guide on the process they followed(!), source: `https://www.adacore.com/press/ada-and-spark-enter-the-automotive-iso-26262-market-with-nvidia`.

And more recently, during DEFCON 33, it was disclosed that NVIDIA has shipped over a billion RISC-V cores all powered by Ada/SPARK, source: `https://www.youtube.com/watch?v=KhWtkZmOPn4`. Wow, just wow.

6 New releases and miscellaneous announcements

Several major Ada packages have seen new releases recently. Namely Alire v2.0, which added several quality-of-life improvements and ARM64 native compilers; Ironclad/GloireOS now run on RISC-V; HiRTOS v2, Muen v1.1 and GHDL v5 were released; and VulkAda added support for Vulkan 1.4!

7 Source material

The slides and recording of the presentation can be found in the FOSDEM website, see `https://archive.fosdem.org/2025/schedule/track/ada/`.

References

[1] AdaCore, "GNAT-LLVM compiler." Available in `https://github.com/AdaCore/gnat-llvm`.

[2] "WebAssembly Core Specification." Available in `https://webassembly.github.io/spec/core/_download/WebAssembly.pdf`.

[3] V. Godunko, "AdaWebPack." Available in `https://github.com/godunko/adawebpack`.

[4] G. Hazebrouk, "G-NAV web." Available in `https://github.com/GuillermoHazebrouck/gnav-web`.

[5] AdaCore, "SPARK Reference Manual." Reference available in `https://docs.adacore.com/spark2014-docs/html/lrm/`.

[6] AdaCore, "SPARKlib, a formally verified library." Available in `https://github.com/AdaCore/SPARKlib`.

Get Started with Ada in 2 Minutes or Less

A. J. Ianozi
email: aj@ianozi.com

Abstract

There are several arguments against Ada surrounding lack of resources for the language and difficulty in tooling, especially in the wake of other safety-focused languages ballooning and surpassing Ada in popularity. Despite old sentiments, getting started with Ada has never been easier thanks to modern Ada tooling such as Alire and GetAda.

Keywords: Alire, Installer, Open Source.

1 Introduction

There is a common sentiment that installing Ada's tooling is difficult, but that isn't the case. One can start with nothing Ada related on their computer to compiling “Hello World” in less than 2 minutes using "GetAda" -- the unofficial installer for Ada's package manager, Alire.

2 Arguments for Rust vs Ada

Despite Ada’s heavy focus on safety-critical software, another language focused on memory safety – Rust – has skyrocketed in popularity over the last several years while Ada’s has increased much more modestly. The prominent arguments identified in the last decade against Ada vs Rust were heavily around barrier of entry, tooling, documentation, and community.

2.1 Low barrier of entry

Rust’s toolchain is very easy to install. By simply running the following command[1] in your terminal, Rustup will automatically install rust’s toolchain on your system:

```
curl --proto '=https' --tlsv1.2 -sSf \
 -sSf https://sh.rustup.rs | sh
```

Newcomers to Ada were presented with a several confusing licenses to choose from and would often post online asking for clarification between GNAT Pro, FSF, or Community Edition[2]. Once an adequately licensed compiler is found, it must still be manually installed on the system.

2.2 Tooling and dependency management

Creating and compiling a new project in Rust is as simple as running the following command:

```
cargo new foo; cd foo; cargo build
```

Rust users have access to hundreds of thousands of libraries on crates.io that can be included into any project by simply issuing the command: `cargo add foo`

2.3 Documentation and Community

Rust has accessible and user friendly documentation[3] that can be leveraged even by absolute beginners along with a large and growing community reaching across communication platforms. The Rust Programming Language Forum[4] has over 30,000 members while Rust’s discord server[5] is even larger with over 57,000 members.

3 Ada’s transition to modern tooling

Ada already has an answer to many of these arguments.

Much effort has been poured into expanding the Ada community, with the Ada Forum[6] reaching over 400 members and Ada’s Discord Server[7] just over 200, and learn.adacore.com is considered one of the best resources in the community for learning Ada.

In 2018, a cargo-like package manager called Alire[8] was announced. Alire serves almost all of the functions of Rust’s cargo. Creating and building a new project in Ada is as simple as issuing:

```
alr init -bin foo; cd foo; alr build
```

Adding libraries is straightforward too, with over 500 crates available on alire.ada.dev/crates. Dependencies can be added by issuing the command: `alr with foo`

In 2022, AdaCore announced[9] the retirement of GNAT Community Edition, allowing Alire to provide the FSF-version of GNAT solving much of the confusing licensing situation.

Despite Alire lowering the barrier of entry for Ada, many recent guides still fail to mention the software and top results for internet searches on downloading Ada still direct people to Community Edition[10]. For those that find Alire, the installation process is very manual on non-Windows systems, requiring extracting binaries and modifying `$PATH`, sometimes even requiring obscure commands if extracted on MacOS[11].

4 GetAda

In an effort to unify Ada’s tooling experience and lower the barrier of entry, GetAda[12] was created to parallel Rustup thus providing a “one-liner” copy-paste terminal command to have a fully working Ada system in seconds.

GetAda is open source and entirely written in Ada. Installing the software is as simple as executing the following in your terminal:

```
curl --proto '=https' \
-sSf https://www.getada.dev/init.sh | sh
```

Once this simple init script is executed, it will automatically download and run GetAda, which in turn will download the latest version of Alire for the target platform and extract it into a path via zipada. GetAda will then detect available shells and update the relevant environment files to ensure Alire's binary is included in $PATH.

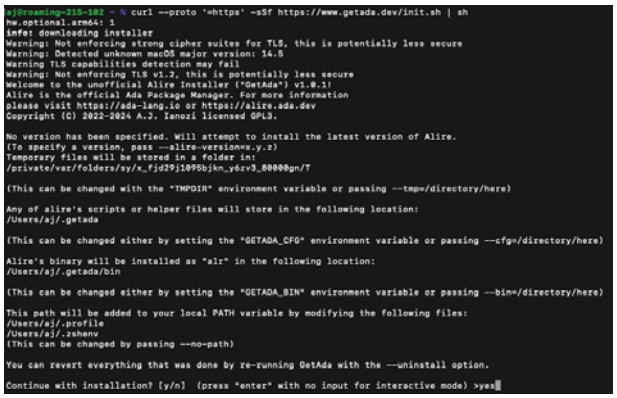


Figure 1: GetAda in action

GetAda will also test the newly-extracted Alire binary to confirm that it works on the target system, and has built-in troubleshooting such as automatically applying chmod +x or running xattr -d com.apple.quarantine. GetAda may be uninstalled at any time by running: getada --uninstall

The GetAda.dev website also provides a quick-start; leveraging GetAda and Alire, one can start with nothing Ada related on their computer to compiling a new project in less than two minutes, simply by issuing the following commands:

```
# Install Alire
curl --proto '=https' \
-sSf https://www.getada.dev/init.sh | sh
# create a new ada project
alr init --bin new_project; cd new_project
# Add a dependency, for example, vss
alr with vss
# Edit your project
alr edit
# Compile your project
alr build
```

5 Conclusion

Version 1.0 of GetAda was released on github[14] in 2024 to incredibly positive reception. It accumulated almost 200 points and over 100 comments on Hacker News[15], and finally was presented at FOSDEM 2025[16].

References

[1] Rustup, https://rustup.rs, 2025

[2] Reddit, "[HELP] Trying to understand the ada ecosystem for commercial purposes." *https://www.reddit.com/r/ada/comments/pk9ne7/help_trying_to_understand_the_ada_ecosystem_for/*, 2021.

[3] The Rust Team, "The Rust Programming Language." *https://doc.rust-lang.org/book/*, 2025.

[4] The Rust Team, "About," The Rust Programming Language Form, *https://users.rust-lang.org/about*, 2025.

[5] Discord, "Invitation to the Rust Discord Server." *https://discord.com/invite/rust-lang*, 2025

[6] Ada-Lang.io. "About," Ada Forum, *https://forum.ada-lang.io/about*, 2025.

[7] Discord, "Invitation to the Ada Discord Server." *https://discord.com/invite/edM2czVKN4*, 2025

[8] A. R. Mosteo, "Alire: a library repository manager for the open source Ada ecosystem.," Ada User Journal, vol. 39, no. 3, 2018.

[9] F. Chouteau, "A New Era For Ada/SPARK Open Source Community.," AdaCore blog, *https://blog.adacore.com/a-new-era-for-ada-spark-open-source-community*, 2022.

[10] AdaCore, "Get started," AdaCore Website, *https:// www.adacore.com/get-started*, 2025.

[11] Alire, "Installation", Alire Docs, *https://alire.ada.dev/docs/#installation*, 2025.

[12] Getada.dev, *https://www.getada.dev*, 2025.

[13] Getada.dev, "How to use Alire?" GetAda, *https://www.getada.dev/how-to-use-alire.html*, 2024

[14] A. J. Ianozi, "GetAda: An unofficial Ada installer." Github, *https://github.com/AJ-Ianozi/getada*, 2025.

[15] A. J. Ianozi, "Show HN: Getada: rustup-like installer for Ada's toolchain/package manager." Hacker News, *https://news.ycombinator.com/item?id=40132373*, 2024.

[16] A. J. Ianozi, "Get started with Ada in 2 minutes or less!", FOSDEM 2025, *https://archive.fosdem.org/2025/schedule/event/fosdem-2025-5056-get-started-with-ada-in-2-minutes-or-less-/*, 2025.

Advent of Compression: Writing a Working BZip2 Encoder in Ada from Scratch in a Few Days

Gautier de Montmollin
Ada Switzerland; email: krikos@bluewin.ch

Abstract

A presentation about a new BZip2 encoder written in Ada was given at the FOSDEM 2025 conference in Brussels. We summarize here its key aspects.

Keywords: Zip-Ada, BZip2, compression, machine learning.

BZip2 is a famous and powerful lossless data compression format. It is very good on some data, like English novels or computer source code. An outstanding aspect of BZip2 is its simplicity. No tricky hash trees or convoluted state machines. However, compression software is generally very difficult to debug, in part impossible. So it is better to do things right the first time and use a language that does its best finding errors before you even run your program.

We showed in the FOSDEM Ada DevRoom how we built a working encoder in a few days by leveraging the precise and customizable type system in Ada, especially range types that seem to have been made for this kind of software.

The real, unexpected, outcome of this short journey is that the new encoder compresses given blocks of data significantly better than other open-source BZip2 implementations, notably the original program, also included in the Info-Zip "zip" tool, and its variant available in the 7-Zip software.

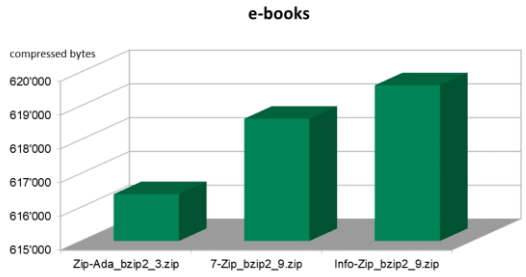


Figure 1. Compressed size of the e-books in the Calgary and Canterbury benchmarks.

That outcome is actually not completely fortuitous. If your principal developing tool lets you describe very precisely the sizes and dimensions of what you want to program, and at the same time takes care of machine-level details like the memory allocation of data-dependent types or the way parameters are passed, you can focus on the algorithms. This is exactly what has lead quickly, in our case, to a better compression scheme for the BZip2 format.

The only equivocal step in the BZip2 encoding is the construction of up to six Huffman trees, whose amount and contents are completely free. Moreover, each bunch of processed data can freely use one of those trees for being entropy-compressed. So, the obvious method for making the best usage of the trees is to simulate for each bunch of data an output with each of the trees and choose the tree leading to the shortest output. Then, the trees may have to be redefined using the statistics of the new selection of data. Since the trees may have changed, you may want to simulate again the encoding and choose again the optimal tree for each bunch of data. This leads to an iterative algorithm which converges in a small number of steps to a stable optimum. It is actually a classic machine learning process, of the clustering family. However, the *initial* setup is the key for the quality of the result: depending on the initial configuration, the algorithm will may be stuck in a suboptimal local optimum. The original BZip2 algorithm is, regarding the initial Huffman tree allocation, trapped in a macro-assembler and bit-masking mindset. The initial trees don't correspond to any subset of the data. The hope is that the iterative process will improve things magically. This works only in part: after the first step, Huffman trees are always corresponding to real data, and there is a convergence.

Our approach to the problem started by plotting the data and finding a way of sorting the bunches of data for getting a meaningful initial tree allocation. So, the initial trees are already computed using data frequencies for selected data, and the data using a certain tree is already approximately homogeneous.

The new BZip2 encoder has made quickly (a couple of weeks after inception) its way as a supported format in the Zip-Ada library (*https://alire.ada.dev/crates/zipada*). This exceptionally short time is in part due to the simplicity of the BZip2 format, but also to the robustness of Ada and the availability of the GNAT toolset. Compression software usually takes a much longer time to reach some kind of stability – it is even common to find trivial out-of-range bugs in C open-source compression code after decades!

Ada and Mini-Ada: a solution to the two-language problem

Gautier de Montmollin
Ada Switzerland; email: krikos@bluewin.ch

Abstract

A presentation about possibilities for avoiding developing software with two different languages was given at the FOSDEM 2025 conference in Brussels. We summarize here its key aspects.

Keywords: GNAT, HAC.

The software world has quickly developed traditions. One of them is to develop prototypes in an easy, readable scripting language X, then to translate them into another language Y which is fast but tricky - typically with a cryptic syntax and fragile memory handling. Alternatively, you see projects with a part in language X and another part in language Y maintained by separate teams, plus people in the middle maintaining the interface between X and Y. Also, some applications in Y embed language X for "live" plug-ins that can be changed during run-time by people who don't develop in Y.

Does it need to be that way?

Actually, Ada makes those traditional but clumsy use-cases with two languages obsolete, sometimes with the help of HAC (the HAC Ada Compiler).

I have shown a modification of the third use-case in a single-language setup, including with a live demo of HAC embedded within a "full Ada" application.

HAC provides a data exchange mechanism with its host application. This mechanism is inspired from Lua, but with type safety and the richer type system that HAC offers (a subset of Ada's type system). In a nutshell, you give the Import aspect to a callback procedure in the Ada application compiled by HAC, and you register the corresponding procedure in the host Ada application that is compiled by a "full Ada" compiler (for instance, GNAT).

In the demonstration, the HAC application is compiled, built and run every three seconds by the host application. The HAC program can be modified freely while the host is running. The effects of the modifications are shown "live" by the host application.

In a real usage, the HAC program could play the role of a "smart" configuration file with some programmatic contents that users can modify while the main application (for instance a game, a user interface with scripted plug-ins, or an embedded applicaton) is running.

For more about HAC:

https://alire.ada.dev/crates/hac
https://hacadacompiler.sourceforge.io/



Understanding Liquid Types, Contracts and Formal Verification with Ada/SPARK

Fernando Oleo Blanco
Open Source & Ada aficionado; Tel: +34 689 44 27 45; email: irvise@irvise.xyz

Abstract

In C-like languages (C++, Rust, Go, Java, C#...) data is generally boiled down to one identifier (variable name) and the computer representation of the data (floats, ints, array, structs...). This method completely devoids the data from its actual, real world meaning and prevents reasoning and analysis beyond some of the most basic properties. In Ada and SPARK, data types are modeled as close as possible to their real world meaning. However, this is generally not enough, as types and the subprograms that operate on them need to be able to represent the nature and properties of the data beyond what is offered by the basic constructs available in Ada. In Ada 2012, type and subprogram annotations became available, allowing the programmer to describe the full set of properties of the data types. This allows programs to be much more accurate, less error prone and easier to understand and reason about. This technique is commonly known in other circles as liquid types and contracts.

The following article will summarize the main ideas and concepts presented in the FOSDEM 2025 talk of the same name as the article.

Keywords: SPARK, contracts, types, liquid types, data modeling

1 Types and data

Since the first few programming languages, the representation of data in computers has generally followed the same structure: an identifier with which to refer to the data and the actual data type. Data types historically have been machine dependent, are tied to the machine representation and hold no real world meaning. Examples of this are Fortran's `REAL*8, LOGICAL...`, C/C++'s `int, char*, [], struct...`, Java, C#... and newer programming languages, such as Go, Rust [1], Zig, Jai... Rust and newer programming languages *fix* the vagueness of C/C++ types by going back to a similar Fortran style types. Therefore, an integer in Rust becomes `u8, u32, s32...` The difference here is that Fortran listed the amount of bytes the type would take, while newer programming languages indicate the amount of bits. It could be said that there has been no real progress in the majority of *mainstream* programming languages since 1962 with FORTRAN IV.

The real tragedy here comes when someone asks a simple question: *what does this `int` represent?* We cannot know. Maybe the variable name that is being declared as an `int` could give us a hint, but still, we cannot know. This is so bad that we cannot even infer that this theoretical *int* refers to a 32-bit signed integer, as that depends on the architecture that we are targeting. For example, in an embedded system, this `int` may only be able to hold 16 bits of signed integer values, as that is the minimum required by the C/C++ standards .

1.1 Pascal, Ada and data modeling

As most of the readers will know, Ada, and one[1] of its parent languages, Pascal, have focused on modeling the real world data with its type system. Instead of focusing on the amount of bits or what kind of registers/assembly are needed for our data, we just declare the data as is. For example `type Floor is range -2 .. 10;` and `subtype Basement is Floor range -2 .. -1;`. These definitions allow us, the compiler and the proof checker to have a much better understanding of the nature of the data and how and what can be done with it. It also serves as self-documenting code, which allows it to be more readable while not having the risk of allowing the in-code documentation to become stale or incorrect.

In Rust and other languages, such as modern C++, types are also used to define correct program flow and force checks where needed. One such example is Rust's `Optional<T>` type, which is the result of the union of the two most common possibilities that may take place during the execution of code. Either the code returns a result, which is represented by a `Some<T>` type or nothing is returned, either because it couldn't or because there was an error, which is represented by a `None`. The compiler will always force the programmer to deal with those two possibilities when accessing the data or when calling functions that return this type. This way, the Rust programming language disallows the "forgetfulness" of checking for errors or different possible data or program states.

2 Logically qualified types, aka, liquid types

With the previous description and uses of types, it is pretty clear that types which describe the real world data or the real state of programs are incredibly useful. However, there are more complex pieces of data that cannot simply be described

[1] Another one being Algol

with a `range` or `Optional<T>` types. One such example are even and odd numbers. When we use these terms, most people understand what they are: the sequence of (positive integer) numbers that are divisible by two or those that aren't respectively. A simple range or union of types cannot express this... logic...

Then, a natural progression of types when presented with the problem above is... to add logic to types. Types that have some kind of logic attached to their definition are know as *logically qualified types*. In some circles, in order to shorten this expression, the short form of *liquid types* is more commonly used. Several languages, most noticeably Haskell, Idris [2], proof related languages such as Rocq [3] and Isabelle [4] and, of course, Ada; implement the functionality to add logical parts to the definition of types.

Therefore, the complete definition of even and odd types in Ada becomes

```
type Even is new Natural with
   Dynamic_Predicate => Even mod 2 = 0;

type Odd is new Natural with
   Dynamic_Predicate => Odd not in Even;
```

We can create substantially more complex logical expression than those shown above. For example, the Prime numbers in Ada can be defined as

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

-- Or more complex data structures
type Day_Temperature is record
   High, Current, Low : Temperature;
end record
   with Dynamic_Predicate =>
     Day_Temperature.High >= Day_Temperature.Current
     and then
     Day_Temperature.Current >= Day_Temperature.Low;

-- Are we sure it is sorted?
type Should_Be_Sorted is array (Index) of Integer;

type Increasing_Ordered_Array is array (Index) of Integer
with Dynamic_Predicate =>
(for all I in Index => (if I < Index'Last then
Increasing_Ordered_Array(I) < Increasing_Ordered_Array(I+1)));
-- Nice!
```

In the aspect `Dynamic_Predicate` we can use any arbitrary function (it must return a Boolean and it must be Pure) which defines the nature of our data. This means that the typical `Valid(My_Data)` function calls, can all disappear if we integrate the `Valid()` function call as part of our type definition. Additionally to the `Dynamic_Predicate` aspect, Ada also offers `Static_Predicate` and `Type_Invariat` aspects, which allow the programmer to refine the nature of the logic that will be used to validate the nature of the data.

2.1 But... Why?

Creating type definitions that hold the full meaning and properties of the data that our program will have to work with, has several advantages with little to no disadvantages:

- Clearer intent and documentation of our data: again, as we are adding more information, **at the source code level** about our data, we help any reader or reviewer understand the full nature of our data.

- More compact code: if we create liquid types, validation procedures, explicit checks, etc; all become unnecessary. This leads to more compact and readable code that focuses on problem solving rather than sanitation.
- Instrumentation: liquids types will disallow incorrect data at runtime if checks are enabled. This allows programmers to very very easily instrument their code to catch quite difficult bugs and to quickly find the root cause.
- Proofs: when we declare liquid types, the provers have a much clearer view and understanding of what is valid and erroneous behaviour. Therefore, liquid types help proof the code in a much more concise and complete way.

A small disadvantage of liquid types in Ada is the potential introduction of recursive calls when doing type conversions. However, this issue can be solved easily by first casting the constrained (read liquid) type to a basic/generic/unconstrained type and then back to a constrained one.

3 Contracts

With liquid types we have seen that there is a way to describe the properties and logic of data. Any cunning person may then have the following reasonable question: *Can we do the same to the executable part of the program, aka, functions and procedures?* After all, any and all programs are black boxes that take some inputs (data), process them (execution) and generate outputs (more data). Therefore, it makes sense to want to extend this *liquidness* to the execution and, in that way, have a full, end-to-end description of the properties of a program, both at the data level and the program level. Well... the answer to the question above is, yes! Adding or listing the properties of executable code is generally known as *functional contracts*. Below, we have a simple and small example of contracts describing the basic properties of procedures that operate on a fictitious *Stack*.

```
procedure Push (V : Character)
   with Pre => not Full,
   -- No need to check in the body
   Post => Size = Size'Old + 1;
   -- Whatever the implementation is, this must hold
procedure Pop (V : out Character)
   with Pre => not Empty,
   -- No need to check in the body
   Post => Size = Size'Old – 1;
   -- Whatever the implementation is, this must hold
procedure Clear
   with Post => Size = 0;
   -- Whatever the implementation is, this must hold
function Top return Character
   with Post => Top'Result = Tab(Last);
   -- Whatever the implementation is, this must hold
```

As we can see in the example above, we add the `Pre`*conditions* and `Post`*conditions* that must hold true for

any Stack to behave correctly. The above example is taken from a heavily simplified version of what is present in the `Platinum_Reusable_Stack` repository [5], which contains several examples of increasing correctness of a Stack.

This contract annotations can be done at the function level, packages level (which could be the Stack example shown) and at the full program level. The programmer can selectively decide the scope for which this technique is applied. Of course, the ideal scenario would be to have the complete program annotated. Doing so, would bring the exact same advantages as described for liquid-types to the full program.

4 Proofs

Modern day provers and, in the case of Ada, SPARK [6]; allow us to systematically analyze the code and understand if, how and where it may fail. Provers utilise the information of the base programming language (design), the liquid types (if any), contracts (if any), the program flow and the general behavioural bounds of computation to answer these questions. Some provers are fully automatic, such as SPARK; and other require some manual guidance, such as Rocq/Coq. Liquid types and contracts are the two most powerful means and methods with which provers can have much more accurate and exhaustive analysis of the programs written. With that added information, they can better analyse the intent, nature and objective of the code that we write, helping us achieve the desired target of zero bugs in the programs we write. However, the explanation on how they are able to do such feat, goes beyond the scope of this paper. Ah! and this is all achieved by the prover even before compiling and running the code!

Additionaly, in the case of SPARK, it has a wonderful little tool which allows the user to have a step-by-step explanation on how something (liquid-type annotation, contracts, etc) may fail during the execution of the code: counter examples. In order to tell GNATprove (the executable that checks the Ada using the SPARK reference), run it with either `-counterexamples=on` or with a higher analysis level `-level=2` or higher level (up-to 4).

5 Source material

The slides and recording of the presentation can be found in the FOSDEM website, see `https://archive.fosdem.org/2025/schedule/track/ada/`.

References

[1] S. Klabnik and C. Nichols, *The Rust Programming Language*. USA: No Starch Press, 2018.

[2] E. Brady, “Idris 2: Quantitative Type Theory in Practice,” pp. 32460–33960, 2021.

[3] T. C. D. Team, “The coq proof assistant,” 2024.

[4] M. W. Tobias Nipkow, Lawrence C. Paulson, “A Proof Assistant for Higher-Order Logic,” pp. 32460–33960, 2002.

[5] AdaCore, “Platinum reusable stack.” Available in `https://github.com/AdaCore/Platinum_Reusable_Stack`.

[6] AdaCore, “Spark reference manual.” Reference available in `https://docs.adacore.com/spark2014-docs/html/lrm/`.

The State of Rust Trying to Catch Up with Ada

Oli Scherer

Rust compiler maintainer; email: mail@oli-obk.de

Abstract

Ada has held its own in the safety critical space for just over 4 decades. Over the last 10 years Rust has pushed into the same space with varying success. Among other features, most notably Rust is missing the powerful declarations and convenience of use of subtypes. In this talk at the FOSDEM 2025 Ada Developer Room I went into the various features that make Ada so useful for ensuring the absence of bugs, and where Rust stands on each of these features.

1 Introduction

At the start of my PhD I was introduced to Ada and subsequently thoroughly disillusioned about my C++ expertise. Over the course of several years working on industry projects in Ada, I examined programming languages and their tooling for concepts to avoid failure states. Though initially scared off Rust, I ended up trying (and failing) to replicate basic Ada Integer subtypes in the Rust type system. I spent the last 10 years working on the Rust language and compiler on improving the compile time evaluation system and recently started adding full on subtypes (called pattern types in Rust).

2 Overview of talk

The talk was not about Rust evangelism nor language bashing.

It was about:

- unsafe/unchecked;
- generic packages/modules;
- safety certification;
- contracts;
- subtypes.

On everything else I think there's little difference in capabilities, even if there are in various important aspects of the usability and error avoidance.

3 More information

Video and PDF transcript of presentation:

https://archive.fosdem.org/2025/schedule/event/fosdem-2025-5356-the-state-of-rust-trying-to-catch-up-with-ada/

Big Integers for Cryptography in SPARK

César Sagaert
AdaCore; email: sagaert@adacore.com

Abstract

Constant-time big integer operations are a critical component of modern cryptography implementations. I present a (partially) formally verified implementation of constant-time big integer arithmetic in SPARK, including modular arithmetic.

1 Constant-time cryptography

Constant time cryptography primitives are an essential part of modern cryptography code. My goal with the `bigints` library [1] was to provide a set of strong primitives for the Ada (and SPARK) ecosystem. It is a fixed-size big integer library in SPARK, with a design and various algorithms heavily inspired by RustCrypto's `crypto-bigint` crate [2].

Despite my best efforts, it is still unaudited cryptographic code written by a single person. I do not advise using it for any kind of security critical application in this state.

All of the subprograms of the library execute in constant time in regard to their inputs, unless explicitly named with a `_Vartime` suffix. That means no branching and no memory accesses that would depend on an input value. Branch timing and power differences, branch prediction, and cache state can and will all leak data about potentially secret values. Constant time does not mean "exactly the same time", but rather always the same instructions and always the same number of cycles.

For instance, in non-constant time binary exponentiation, power consumption analysis can lead to the entire RSA secret exponent being leaked. A naive binary exponentiation would branch on each bit of the secret exponent and perform, for a 0, a modular squaring, and for a 1, a modular squaring followed by a modular multiplication. Those two operations do not have the same cost for the CPU, and an attacker could theoretically just read the exponent from a power consumption graph.

What we can do instead is always perform the two operations, and select the correct result in constant time using clever masking tricks, like this "conditional select" primitive:

```
f(A, B, mask) = A ^ (mask & (A ^ B))
```

This makes the code a bit slower, and usually less readable. Additionally, a good optimizing compiler might recognize your constant time algorithm and replace it with a faster, variable time one!

2 Proving contracts in SPARK

Choosing SPARK for this project has various advantages. The first being the guaranteed absence of runtime errors. We can also compensate the lost visibility of our algorithms by formally proving our constant-time primitives:

```
function Cond_Select (
   A, B : Uint; C : Choice
) return Uint with Post =>
   Cond_Select'Result = (if To_Bool (C)
                         then B else A);
```

However, we cannot prove that these operations are effectively in constant time, as GNATprove has no notion of the concept. All constant time code that is not written in raw assembly relies on *best effort* implementations; some techniques involve obfuscating intent from the compiler, and adding optimization barriers where needed. Convincing ourselves that our code is effectively in constant time can be done by auditing assembly output, and measuring execution time with statistics on cycles, power draw...

3 Higher level proofs?

Beyond basic SPARK niceties, we would like to add functional proofs: this can be done using Ada's `Big_Integer` package in SPARK "ghost" code – annotated code that never ends up in the compiled objects, and is used exclusively for proofs. For instance, a contract on our big integer addition could look like:

```
function Add (A, B : Uint) return Uint
with Post =>
   F (A) + F (B) = F (Add'Result);
```

with `F` being an evaluator function converting our `Uint` to Ada's `Big_Integer` type. The `Uint` type is basically a fixed size array of 64-bit "limbs", and can be understood as the representation of the integer in base `2**64`.

The addition proof can be done by induction, when adding the two integers limb by limb with an extra word for the carry. It requires a lot more ghost code than actual runtime code, as is usually the case with complex proofs; it is however very exciting to see how far we can push GNATprove to prove functional contracts.

In the `bigints` repository, the proof for the addition function is available in the `addition-proof` branch, because at the time of writing it, the precompiled GNATprove available in the Alire index was not able to prove it.

4 Additional work

This was the first (and for now, only) personal project I contributed to the Alire index.

Writing this library led me to write an ad-hoc test suite, which was the basis for the test suite capabilities I contributed to Alire later on [3].

This test suite is not only for correctness control, it also contains a little showcase of the library's modular arithmetic capabilities, with a small elliptic curve implementation [4]. See the `tests/src/tests-curve25519.adb` file in the repository for details.

References

[1] "Constant time big integers in spark." `https://github.com/AldanTanneo/bigints`.

[2] "Rustcrypto: Cryptographic big integers." `https://github.com/RustCrypto/crypto-bigint`.

[3] "Proposal for integrated test suite in alire." `https://github.com/alire-project/alire/issues/1831`.

[4] "Elliptic curves for security." `https://www.rfc-editor.org/rfc/rfc7748`.

Multiword Arithmetic and Parallel Computing

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

In many applications, the precision by the available hardware arithmetic is insufficient to guarantee accurate results. Multiword arithmetic is a special type of multiprecision arithmetic where a multiple double is an unevaluated sum of 64-bit doubles, or where a multiple integer is an unevaluated sum of 64-bit integers. Parallel computing is applied to compensate for the cost overhead of multiword arithmetic. This type of arithmetic exploits naturally the optimized hardware, allows for efficient type conversions, memory layouts, all favorable for parallel computing. For example, storing a multiword in registers rather than arrays is beneficial to parallel computing by tasking and acceleration by graphics processing units. Code for multiword arithmetic is available in the software PHCpack, written mainly in Ada, publicly available at github, and as an Alire crate, released under the GNU GPL v3.0 license.

1 Introduction

An algorithm is *robust* if it does not fail for small perturbations of degenerate inputs [1]. Software can be made more robust by the application of multiprecision arithmetic. The algorithms [2] to extend 32-bit floating-point arithmetic using originated in the late sixties [3] and available software packages are QDlib [4] and CAMPARY [5].

A *multiple double* is an unevaluated sum of nonoverlapping doubles. The multiple double arithmetic multiplies the accuracy of the results, from 16 decimal places to 32 and 64, respectively for double doubles and quad doubles. However, the cost overhead is significant, and parallel computing allows to compensate for this cost overhead. The overhead factors are summarized in Table 1.

	`add`	`mul`	`div`	`avg`
2	20	23	70	37.7
4	89	336	893	439.3
8	269	1742	5126	2379.0
16	925	11499	33041	15155.0

Table 1: The number of operations with doubles for a multiple double addition (`add`), multiplication (`mul`), and division (`div`), required for m-double arithmetic, for $m = 2, 4, 8, 16$.

*Supported by a 2023 Simons Travel Award.

This paper describes some recent additions to PHCpack [6], extending contributions described in [7] and [8]. One application is the computation of power series [9]. As the errors in the leading coefficients propagate to the trailing coefficients, the leading coefficients must be computed at higher accuracy than what can be computed in 64-bit double precision.

The code described in this paper serves as a computational preparation for use of tensor cores on a graphic processing unit, in a stepping stone to implement the Ozaki scheme [10].

2 An Error Free Summation

Assuming all 64-bit doubles have the same exponent, we work with 52-bit integers (fractions of the doubles). The idea for an error free summation is introduced in Figure 1.

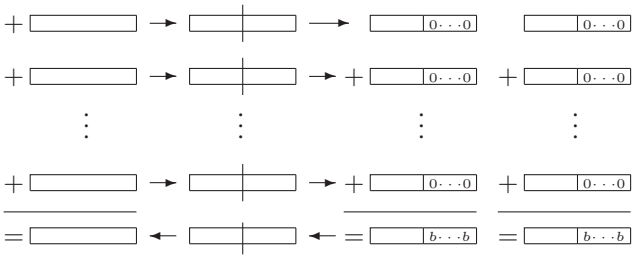


Figure 1: Split a vector of doubles, add the parts, fuse the result.

If the number of additions does not exceed some threshold, then we have sufficiently many zero bits left at the end of the numbers to represent the result exactly, without any error.

The idea in Figure 1 will be applied to computing inner products with double double arithmetic. Given are vectors $\mathbf{x}$ and $\mathbf{y}$ both of length n, of double double numbers, we compute $\sum_{k=1}^{n} x_k \star y_k$, where $\star$ is the double double multiplication.

The double double x_k is represented by $(x_k^{\text{hi}}, x_k^{\text{lo}})$, where the high double x_k^{hi} and the low double x_k^{lo} of x_k are splitted in quarters:

$$(\overbrace{x_{k,0}, x_{k,1}, x_{k,2}, x_{k,3}}^{x_k^{\text{hi}}}, \overbrace{x_{k,4}, x_{k,5}, x_{k,6}, x_{k,7}}^{x_k^{\text{lo}}}).$$

After splitting also y_k, we compute in double arithmetic:

$$s_0 = \sum_{k=1}^{n} x_{k,0} y_{k,0}, \;\; s_1 = \sum_{k=1}^{n} x_{k,1} y_{k,0} + x_{k,0} y_{k,1},$$

and

$$s_i = \sum_{k=1}^{n} \sum_{j=0}^{i} x_{k,j} y_{k,i-j},$$

for $i = 2, \ldots, 7$. Then, add $s_0 + s_1 + \cdots + s_7$ in double double arithmetic.

To examine the computational efficiency, random 64-bit doubles are generated with a fraction of 52 bits in following pattern:

$$1\underbrace{bb\cdots b}_{12\text{ bits}}1\underbrace{bb\cdots b}_{12\text{ bits}}1\underbrace{bb\cdots b}_{12\text{ bits}}1\underbrace{bb\cdots b}_{12\text{ bits}}, \quad b \in \{0,1\}.$$

Splitting such double into four leads to doubles with fractions

$$\begin{aligned}
&1b\cdots b\ 00\cdots 0\ 00\cdots 0\ 00\cdots 0,\\
&00\cdots 0\ 1b\cdots b\ 00\cdots 0\ 00\cdots 0,\\
&00\cdots 0\ 00\cdots 0\ 1b\cdots b\ 00\cdots 0,\\
&00\cdots 0\ 00\cdots 0\ 00\cdots 0\ 1b\cdots b.
\end{aligned}$$

By virtue of the placement of the ones in the random fractions, all quarters have fixed exponents, e.g.: 0, -13, -26, -39. All doubles in a multiple double are generated according this pattern.

3 Computational Results

The results on computing 1,024 times $\sum_{k=1}^{6144} a_k \star b_k$ in increasing precision is shown in Table 2.

	ordinary		speedup	vectorized	
16d	40s 780ms	6.3x	**4.3x**	9s 491ms	6.2x
8d	6s 428ms	3.3x	**4.2x**	1s 520ms	4.8x
4d	1s 977ms	12.x	**6.2x**	318ms	4.6x
2d	158ms	13.x	**2.3x**	69ms	2.3x
1d	12ms		**0.4x**	30ms	

Table 2: Rows for double (1d), double double (2d), quad double (4d), octo double (8d), and hexa double (16d), comparing the new vectorized inner products to the ordinary ones. Times are in milliseconds (ms). The column next to the times are the cost overhead factors.

Times in Table 2 were obtained on an Intel Xeon 5318Y Ice Lake-SP, up to 3.40GHz, 256GB of internal memory at 3200MHz, GNU/Linux, Microway 2024, compiled with GNAT 12.2.0, flags `-O3 -gnatp -gnatf`.

It takes 9 seconds for 1,024 inner products in hexa double precision (16d). The wall clock time is 9s 308ms, with 85ms for generating the vectors. In a high level multithread computation, every thread does one inner product. On two 24-core Intel Xeon 5318Y Ice Lake-SP, up to 3.40GHz, 256GB of internal memory at 3200MHz, GNU/Linux, Microway 2024, compiled with GNAT 12.2.0, flags `-O3 -gnatp -gnatf`, the wall clock time drops to 293 milliseconds, using 96 threads.

Comparing the 293 milliseconds to the 318 milliseconds with one thread in quad double precision, we can quadruple the precision and compute as fast as in quad double precision, using 96 threads, achieving *quality up*.

4 Conclusions

Postponing renormalizations of multiple doubles benefits the efficiency.

The convolutions $\sum_{k=1}^{n} \sum_{j=0}^{i} x_{k,j} y_{k,i-j}$ allow to rewrite the inner products in multiple double arithmetic as matrix multiplications in double precision floating-point arithmetic, to prepare for better acceleration with graphics processing units, in particular tensor cores.

References

[1] J. R. Shewchuk, "Adaptive precision floating-point arithmetic and fast robust geometric predicates," *Discrete Comput. Geom.*, vol. 18, no. 3, pp. 305–363, 1997.

[2] J.-M. Muller, N. Brunie, F. de Dinechin, C.-P. Jeannerod, M. Joldes, V. Lefèvre, G. Melquiond, N. Revol, and S. Torres, *Handbook of Floating-Point Arithmetic*. Springer-Verlag, second ed., 2018.

[3] T. J. Dekker, "A floating-point technique for extending the available precision," *Numerische Mathematik*, vol. 18, no. 3, pp. 224–242, 1971.

[4] Y. Hida, X. S. Li, and D. H. Bailey, "Algorithms for quad-double precision floating point arithmetic," in *15th IEEE Symposium on Computer Arithmetic (Arith-15 2001)*, pp. 155–162, IEEE Computer Society, 2001.

[5] M. Joldes, J.-M. Muller, V. Popescu, and T. W., "CAMPARY: Cuda multiple precision arithmetic library and applications," in *Mathematical Software – ICMS 2016, the 5th International Conference on Mathematical Software*, pp. 232–240, Springer-Verlag, 2016.

[6] J. Verschelde, "Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation," *ACM Trans. Math. Softw.*, vol. 25, no. 2, pp. 251–276, 1999.

[7] J. Verschelde, "Parallel software to offset the cost of higher precision," *ACM SIGAda Ada Letters*, vol. 40, no. 2, pp. 59–64, 2020.

[8] J. Verschelde, "Exporting Ada software to Python and Julia," *ACM SIGAda Ada Letters*, vol. 42, no. 1, pp. 76–78, 2022.

[9] J. Verschelde, "GPU accelerated Newton for Taylor series solutions of polynomial homotopies in multiple double precision," in *Proceedings of the 26th International Workshop on Computer Algebra in Scientific Computing (CASC 2024)* (F. Boulier, C. Mou, T. M. Sadykov, and E. V. Vorozhtsov, eds.), vol. 14938 of *Lecture Notes in Computer Science*, pp. 328–348, Springer-Verlag, 2024.

[10] K. Ozaki, T. Ogita, S. Oishi, and S. M. Rump, "Error-free transformations of matrix multiplication by using fast routines of matrix multiplication and its applications," *Numerical Algorithms*, vol. 59, pp. 95–118, 2012.

Developing Device Drivers for Ironclad Using Ada

Cristian Simon

Ironclad; email: streaksu@mailbox.org; website: https://ironclad-os.org

Abstract

Ironclad is a partially formally verified, hard real-time capable kernel for general-purpose and embedded uses, written in SPARK and Ada. This paper delves into why Ada was chosen as development language, and how Ada is used inside the project with device drivers as an example.

Keywords: operating systems, device drivers, formal verification, SPARK, Ada.

1 Introduction

Ironclad was started as a fully free software [1] hobbyist project to develop a kernel for powering general purpose operating systems, but, with time, broadened scope to the field of formal verification and real-time computing.

The main question among prospective developers trying to approach the project is "why Ada?", portraying Ada as a weird tool to use, a flavor choice by the creator of the project at the expense of accessibility, and overall a hurdle to overcome. Truth is, a lot of the things that make Ironclad special are explicitly possible thanks to Ada.

2 Formal verification as a foundation

Ironclad has as main focus its real time capabilities, scheduling guarantees, and formal verification. Despite these being three separate points, formal verification plays a more foundational role than the other two. The formal verification of Ironclad provides the backing needed to ensure the other two components behave as they should, by ensuring constrains on how security facilities like cryptography and Mandatory Access Control (MAC) are followed, and verifying scheduling guarantees only affect what we intend to affect.

2.1 Ada's formal verification advantage

Having established how foundational formal verification is for the project, one thing is for sure, we need a good formal verification framework.

In this field, Ada is light-years ahead of the competition like C or Rust. SPARK is really on a league of its own when it comes to fully FOSS formal verification. The ability of having the formal verification proofs being able to be embedded on the Ada code instead of using separate verification documents also makes development extremely easy.

3 The challenge of formal verification

SPARK makes formal verification easy with its more comfortable verification process, still, formal verification of an operating system kernel is no small task. The seL4 team [3] reports that "20 person years for 10 000 source lines of C code" [4] are needed. Not only that, but a project like seL4, a microkernel with userland servers providing a big chunk of the kernel utility, has to implement a reduced set of kernel functionality. Ironclad does need to be more complex and extensive, as we have to provide the same tools as seL4 alongside the more complicated POSIX interfaces, drivers, and IPC mechanism. Due to this, we have to make a few choices when it comes to formally verifying Ironclad if we are to ever finish it with our current limited manpower.

3.1 Picking our battles

Not every part of the kernel has the same security and integrity requirements, and not every part of the kernel has to have these requirements applied with the same strength and consistency. In Ironclad we use this principle to place a higher priority in verifying security facilities, which are mostly completely verified to SPARK's gold level [5], to then move to things like Inter-Process Communication (IPC), process management, to then verify architecture-specific code like page management and context switches with the lowest priority. It is usually correlated with how close the code is to userland (See Figure 1).

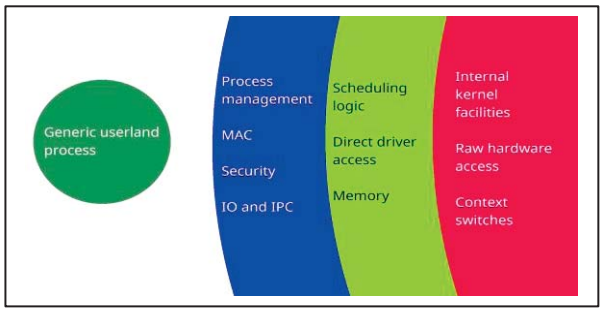


Figure 1. Verification priority in Ironclad

3.2 Tools SPARK gives us

SPARK gives us tools to pick these battles effectively, the main ones it gives us are the ability to enable and disable verification in bodies and specifications by the use of the `SPARK_Mode` attribute, which allows us to use unverified bodies with verified specifications, or separate code into verified and unverified functions, creating conglomerates of modules featuring different levels of verification seamlessly.

4 Ada's quality of life features

High-level languages usually are made with little regard for use on lower level environments like embedded devices or operating system kernels, Ada though handles this environment perfectly. Ada features mature freestanding-capable toolchains, and a specification that leads little room

for idiosyncrasies from the world of baremetal development to ruin your day.

A good example is the handling of bitfields with Ada's representation clauses and extremely flexible types, which allow you to define types down to the bit, and simplify perfectly mapping memory registers into easily accessible representations. Other languages like C feature similar options that are known for being minefields [6], while Rust lacks these features all-together in the language [7], having to rely on second-party crates.

5 Runtime configuration and customization

Another big advantage of Ada is how the runtimes are highly configurable. When compared with other high-level language runtimes that can cause weird chicken-and-egg problems (needing another language without a runtime to bootstrap a runtime's allocator, for example), Ada allows you to pick and choose which parts of the runtime you would like to engage with the use of pragmas (See Figure 2).

```
pragma Restrictions (No_Obsolescent_Features);
pragma Restrictions (No_Elaboration_Code);
pragma Restrictions (No_Floating_Point);
pragma Restrictions (No_Finalization);
pragma Restrictions (No_Secondary_Stack);
```

Figure 2. Example of runtime-restricting pragmas

These restrictions can be applied program-wide or in a scope by scope basis, and can be useful to address a range of issues, like disabling floating point functionality until that functionality is available.

6 Current state, future, and roadmap

Ironclad is featured as the kernel of Gloire, an operating system using it alongside core userland utilities of the GNU [8] project, like coreutils and GCC, and the libc of the managarm [9] project, alongside applications and environments based on Xorg and GTK+3 (See Figure 3).

Gloire, just like Ironclad, is fully free software. Disk images ready for use, along with instruction on how to use them, and other relevant documentation, are available on-line at https://codeberg.org/Ironclad/Gloire.

In the future, we expect to delve on improving networking support for Ironclad and finalize our work on implementing a fully fledged USB stack for the kernel.

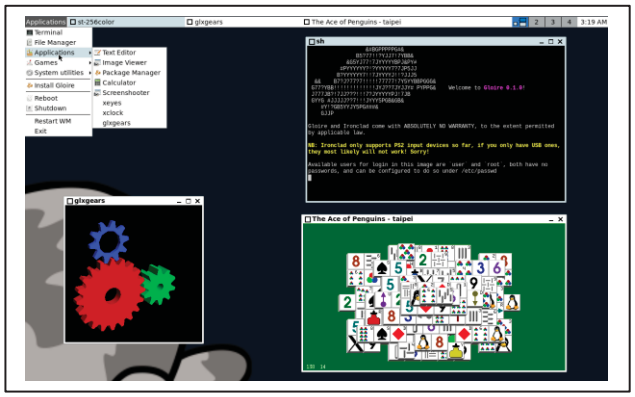


Figure 3. Screenshot of Gloire as of Jun 2025

Ironclad's documentation, as well as download links, presentation materials, contribution information, and RSS feeds to follow development, among others, are available on-line at https://ironclad-os.org.

References

[1] "What is Free Software?" By the Free Software Foundation https://www.gnu.org/philosophy/free-sw.html.en

[2] Gloire: An OS built with the Ironclad kernel and GNU tools https://github.com/streaksu/gloire

[3] seL4: The world's most highly assured and fastest operating system kernel https://sel4.systems/

[4] Don't Sweat the Small Stuff: formal verification of C code without the pain (David Greenaway, Japheth Lim, June Andronick, Gerwin Klein, 2014) https://dl.acm.org/doi/10.1145/2666356.2594296

[5] "Applying SPARK in practice", by AdaCore, part of the SPARK's User Guide, covering assurance levels https://docs.adacore.com/spark2014-docs/html/ug/en/usage_scenarios.html

[6] "Betrayed by a bitfield" – LWN.net article https://lwn.net/Articles/478657/

[7] Rust RFC https://github.com/rust-lang/rfcs/issues/314

[8] "What is GNU?" By the GNU Operating System Project https://www.gnu.org/

[9] The Managarm project's GitHub repository https://github.com/managarm/managarm

Kids Learn to Code with AdaBots

Tama McGlinn
adabots.net; email: hello@adabots.net

Abstract

We present an open source software stack that creates a teaching environment especially well suited to teaching children the Ada programming language. AdaBots are programmable bots in a 3 dimensional voxel world, similar to Minecraft. These are built as a mod for minetest, and listen via a proxy server to an Ada program. This program is a normal Ada executable built with the adabots library, available through Alire, and controls one or more AdaBots with instructions such as `dig`, `turn_right`, `place_down` *with which it can move about and modify the world in minetest. We posit that it is best to learn in a realistic environment, using the same tools real programmers use in the field, so that minimal effort is required later to transition from the learning environment to the real world.*

Keywords: Ada, teaching.

1 Introduction

Most systems for learning to program, especially those geared toward children, are presented as a *novel* form of programming, using graphical reprensetations in lieu of code. This puts the student at a distance from the software that makes it hard to progress beyond preconceived examples to truly complex user programs.

Instead, we created a library to be used by ordinary standalone programs. This library can send instructions to a virtual robot (an *AdaBot*) running inside a minetest world. However, there is nothing special about the programming environment. On adabots.net [6] we offer virtualized and preconfigured VSCodeWeb workspaces, but the AdaBot can equally well listen to any machine running a program using the AdaBots library.

2 Example program

Consider the following example program. Given a bot that is at a cliff, this builds a bridge which the player can walk across. The bot itself can expend fuel to fly, but non-administrators in a typical multiplayer minetest server cannot.

```
with Adabots;
procedure Cross_Gap is
   Bot : constant Adabots.Turtle := Adabots.Create_Turtle;
begin
   loop
      exit when not Bot.forward;
      exit when Bot.Detect_Down;
      Bot.Place_Down;
   end loop;
end Cross_Gap;
```

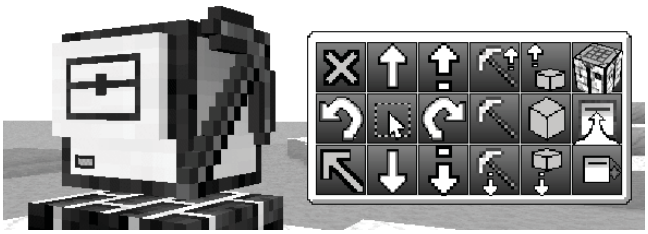


Figure 1: A bot with control panel open

Starting on the first line in the loop, we see that the function `bot.forward` is used, which returns `False` when it hits a wall and cannot proceed. This is to handle the case when the other side of the chasm is higher than the side we started on.

On the second line of the loop, we have moved forward and check if there is ground beneath - if not, we place a block below and continue the loop.

Using such programs, you can explore a minetest world with students and learn about the Ada programming language, and devise construction and mining challenges.

3 Bot capabilities

3.1 Movement

An AdaBot can interact with the nodes immediately in front, above and behind the single node they occupy. Non-air nodes can be mined if a pickaxe is equipped in the bot. If the selected slot of the 16-slot inventory has blocks in it, the bot can place these in the same three locations. For movement, we have the same three directions and also `bot.back` to move back, and `bot.turn_left` and `bot.turn_right`.

By default, the virtualized minetest server you can get on adabots.net is configured so that the bots do not require fuel. But this can be changed during play, so that the bot must find coal or other combustibles and consume them to move, dig, construct and to fly.

3.2 Seeing

The bot can `detect[_Up|_Down]`, which returns `true` iff there is a (non-air) block occupying the node. Using `inspect[_Up|_Down]` yields the actual node-type.

To see further away, there is a function coming soon which yields a cone of nodes in any of the three directions with relative coordinates paired to the node-type.

3.3 Items

The bot can pick up and drop items in the world and also use chests and furnaces. As most minetest servers do not require the crafting table, the AdaBot can also craft directly from its inventory. Hence, it is possible for an AdaBot program, starting with enough fuel and a pickaxe, to mine, smelt, and craft the necessary ingredients to place another AdaBot.

3.4 Management

A control panel in-game, shown in figure 1 on the right, allows the immediate execution of these commands. Another such menu offers direct interchange of the player and bot inventories, shows the fuel state and equipped tool, and allows the user to choose a name. The name is displayed above the bot in the world and can be used to distinguish bots in multi-bot programs.

Administrator accounts are able to teleport bots and teleport to the bots location. The owner of a bot or an administrator can modify the list of players able to interact with the bot via the control panel.

4 Related work

Released in 1986 for the Apple II, Robot Odyssey [9] features robots with custom-wired programmable hardware. For something more recent, see Colobot (2001) [1]. This uses a higher-level programming language called *CBOT*, which bears an uncanny resemblence to C++. In 2017, Autonauts [5] was released, which uses the popular Scratch [7] visual programming system.

As far as we are aware, there are no previous systems for learning programming that decouple the coding environment as we have done, to create a language and editor-agnostic environment.

My own first encounter with programmable bots in a game environment was through ComputerCraft [3], further developed along two separate tracks:

First, in the open-source CC:Tweaked [8], which adds many game features but sticks to the limiting in-game Lua based text editor environment.

Second, in the Microsoft-backed closed source Edu version [4], which introduces a new visual programming language that seems well-suited to children at first but is tedious in use in practice. For example, it is impossible to define procedures or functions.

5 Online virtualized environment

Although a bot will listen to any coding environment, whether local or hosted on a server, we make it easier to get started using AdaBots by hosting VSCodeWeb workspaces which are spawned on a per-student basis using Coder [2].

Similarly, although AdaBots is a free and open-source mod anyone can install onto their own configured minetest server, we offer preconfigured dockerized minetest servers which are accessible either from the minetest client or via our web client.

Together this scalable stack allows clients to use AdaBots with only a browser being required. However, we do recommend clients to connect using multiple screens and preferably two computers, so that teacher and student can modify code and see both minetest and code without needing to switch.

6 Future work

In the future, we hope to extend the system to make it easier to use other programming languages. A simple wrapper around each web call to move the bot is all that is necessary to enable a new programming language, but to properly support this we need to create tutorials for the new language as well.

As for the bots inside minetest, we hope to extend the available behavior in future, with;

- some sort of "cheat-mode", allowing direct manipulation of blocks
- easier ways to find lost bots
- reviving integration with MineCraft
- many more tutorials

However, as the author is using the system also for teaching his children, we do feel the system is good enough as it is, especially if you are an Ada programmer already, and have the time to teach students one on one.

References

[1] Colobot, 2001. URL `https://colobot.info/`.

[2] Coder. Open source self-hosted cloud development environment, 2025. URL `https://coder.com/`.

[3] Daniel Ratcliffe (dan200). Computer craft, 2011. URL `https://github.com/dan200/ComputerCraft/`.

[4] Microsoft Daniel Ratcliffe. Computer craft edu, 2015. URL `https://computercraftedu.com/`.

[5] Denki. Autonauts - build, create, automate, 2017. URL `https://denki.itch.io/autonauts`.

[6] Tama McGlinn. The #1 programming site for children, 2025. URL `https://adabots.net/`.

[7] Mitch Resnick. Scratch: Learn to code, code to learn, 2025. URL `https://scratch.mit.edu/`.

[8] Jonathan Coates (SquidDev). Computer craft tweaked, 2017. URL `https://github.com/cc-tweaked/CC-Tweaked/`.

[9] The Learning Company Mike Wallace and Dr. Leslie Grimm. Robot odyssey, 1984. URL `https://en.wikipedia.org/wiki/Robot_Odyssey`.

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. Johan Nielsen
Email: jon@jonsoft.dk

Ada-Deutschland

Dr. Hubert B. Keller CEO
ci-tec GmbH
Beuthener Str. 16
76139 Karlsruhe
Germany
+491712075269
Email: vorstand@ada-deutschland.de
URL: ada-deutschland.de

Ada User Society

c/o Ahlan Marriott
Altweg 5
8450 Andelfingen
Switzerland
Phone: +41 52 3171975
e-mail: treasurer@ada-user.org
URL: www.ada-user.org

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 3171975
e-mail: president@ada-switzerland.ch
URL: www.ada-switzerland.ch

Ada-Europe Sponsors



46 Rue d'Amsterdam
F-75009 Paris
France
sales@adacore.com
www.adacore.com



506 Royal Road
La Caverne, Vacoas 73310
Republic of Mauritius
Contact: David Sauvage
david.sauvage@adalabs.com
www.adalabs.com



2 rue Mozart
92110 Clichy
France
Contact: Jean-Pierre Rosen
rosen@adalog.fr
www.adalog.fr/en/



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



3271 Valley Centre Drive,Suite 30(
San Diego, CA 92069
USA
Contact: Shawn Fanning
sfanning@ptc.com
www.ptc.com/developer-tool



Enterprise House
Baloo Avenue, Bangor
North Down BT19 7QT
Northern Ireland, UK
enquiries@sysada.co.uk
sysada.co.uk



1115 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





Bachstrasse 51
8200 Schaffhausen
Switzerland
Contact: Ahlan Marriott
admin@white-elephant.ch
www.white-elephant.ch




Ada
europe