ADA USER JOURNAL

Volume 39
Number 1
March 2018

Contents

Editorial Policy for Ada User Journal

Publication

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

Aims

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

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

The Journal publishes the following types of material:

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

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

Original Papers

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

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

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

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

Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.

Proceedings and Special Issues

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

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

News and Product Announcements

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

Reprinted Articles

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

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

Commentaries

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

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

Announcements and Reports

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

Reviews

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

Submission Guidelines

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

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

Editorial

In this Editorial, I would like to start by pointing out the preliminary program of the Ada-Europe 2018 conference, June 18-22, in Lisbon, Portugal, which you can find in the forthcoming events section of the Journal. The conference program includes three very valuable keynotes (on Tuesday, Paulo Esteves-Veríssimo, from the University of Luxembourg, with a keynote about “Security and Dependability Challenges of IT/OT Integration”; on Wednesday, Carl Brandon, from the Vermont Technical College, USA, will provide the perspective “From Physicist to Rocket Scientist, and How to Make a CubeSat that Works”; and Thursday, Erhard Plödereder, from the University of Stuttgart, Germany, will talk about “Vulnerabilities in Safety, Security, and Privacy”), nine sessions of technical papers and industrial presentations, an extensive group of tutorials and two workshops on Monday and Friday. The week will also have an interesting social program – details will be increasingly available in the conference website.

I am looking forward to meeting you all in Lisbon!

And this year is full of Ada-related events. After the Ada Developer Room at FOSDEM, last February, and a new edition of the International Real-Time Ada Workshop, taking place in Benicàssim, Spain, 18-20 April, we will also have the ACM SIGAda High Integrity Language Technology workshop, 5-6 November, in Boston, USA. Information about the latter can also be found in the forthcoming events section.

As for the technical content of the issue, the first article, from a group of authors from the Sfax University, Tunisia and Abdulaziz University, Saudi Arabia, presents an approach using text mining techniques to extract common and variable features from product variants. Afterwards, Victor Porton, from Israel, discusses the experiences with writing an Ada binding to the Redland RDF libraries. Finally, Yannick Moy presents how SPARK can be used to prove a brute force version of string search, and to discover a bug in a faster quick search version.

As usual, the reader will also encounter the information provided in the News Digest and Calendar sections, prepared by Jacob Sparre Andersen and Dirk Craeynest, their respective editors.

Luís Miguel Pinho
Porto
March 2018
Email: AUJ_Editor@Ada-Europe.org

Ada User Journal

Call for Contributions

Topics: **Ada, Programming Languages**, **Software Engineering Issues** and **Reliable Software Technologies** in general.

Contributions: **Refereed Original Articles**, **Invited Papers**, **Proceedings** of workshops and panels and **News and Information** on Ada and reliable software technologies.

More information available on the
Journal web page at

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

Online archive of past issues at http://www.ada-europe.org/auj/archive/

Quarterly News Digest

Jacob Sparre Andersen

Jacob Sparre Andersen Research & Innovation. Email: jacob@jacob-sparre.dk

Contents

Ada-related Organisations

ACM SIGAda 2017 Robert Dewar Award

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri Dec 15 2017
Subject: Dr. Peter Chapin (Vermont Technical College) receives ACM SIGAda 2017 Robert Dewar Award
URL: http://www.adaic.org/2017/12/dr-peter-chapin-vermont-technical-college-receives-acm-sigada-2017-robert-dewar-award/

The ARA congratulates Dr. Peter Chapin on his receipt of ACM SIGAda's Robert Dewar Award, which acknowledges outstanding contributions to the Ada community. Dr. Chapin was a major contributor to the Vermont Tech Lunar CubeSat project (cubesatlab.org) whose software was written in SPARK/Ada. The Vermont Tech CubeSat was launched in November 2013 and successfully completed its full two-year mission, the only one out of twelve academic CubeSats to do so. Dr. Chapin attributes the software's reliability in large part to the SPARK/Ada technology, which was used to prove the absence of run-time errors.

Dr. Chapin is now coordinating the work on CubedOS, a SPARK/Ada implementation of a software framework for small spacecraft, with plans to release the result as an open-source project. Other groups will thus have access to a high-integrity software base for their CubeSats, which currently have a very high failure rate.

Dr. Chapin is the co-author, along with Prof. John McCormick, of “Building High Integrity Applications with SPARK”, a student-oriented textbook on SPARK 2014.

Ada-related Events

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

FOSDEM 2018

From: Dirk Craeynest <dirk@cs.kuleuven.be>
Date: Wed, 10 Jan 2018 22:18:04 -0000
Subject: FOSDEM 2018 - Ada Developer Room - Sat 3 Feb 2018 - Brussels
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

Ada-Belgium is pleased to announce the program for its
8th Ada Developer Room at FOSDEM 2018
on Saturday 3 February 2018

Université Libre de Bruxelles (ULB), Solbosch Campus, Room AW1.125
Avenue Franklin D. Roosevelt Laan 50, B-1050 Brussels, Belgium

Organized in cooperation with Ada-Europe

<http://www.cs.kuleuven.be/~dirk/ada-belgium/events/18/180203-fosdem.html>

<http://fosdem.org/2018/schedule/track/ada/>

FOSDEM, the Free and Open source Software Developers' European Meeting, is a non-commercial two-day weekend event organized early each year in Brussels, Belgium. It is highly developer-oriented and brings together 8000+ participants from all over the world. The goal is to provide open source developers and communities a place to meet with other developers and projects, to be informed about the latest developments in the open source world, to attend interesting talks and presentations on various topics by open source project leaders and committers, and to promote the development and the benefits of open source solutions. The 2018 edition takes place on Saturday 3 and Sunday 4 February. It is free to attend and no registration is necessary.

In this edition, Ada-Belgium organizes once more a series of presentations related to the Ada Programming Language and Free or Open Software in a s.c. Developer Room. The "Ada DevRoom" at FOSDEM 2018 is held on the first day of the event, Saturday 3 February 2018.

Ada Programming Language and Technology

Ada is a general-purpose programming language originally designed for safety- and mission-critical software engineering. It is used extensively in air traffic control, rail transportation, aerospace, nuclear, financial services, medical devices, etc. It is also perfectly suited for open source development.

Awareness of safety and security issues in software systems is increasing. Multi-core platforms are now abundant. These are some of the reasons that the Ada programming language and technology attracts more and more attention, among others due to Ada's support for programming by contract and for multi-core targets. The Ada 2012 language definition was approved and published by ISO in December 2012, updated early 2016, and work on new features for the next revision is ongoing. As with the prior Ada 1995 and Ada 2005 standards, the first full implementation of the Ada 2012 standard was made available in gcc - the GNU Compiler Collection (GNAT). More and more tools are available, many are open source, including for small and recent platforms. Interest keeps increasing, also in the open source community, and many exciting projects started.

The Ada DevRoom aims to present the facilities offered by the Ada language (such as for object-oriented, multicore, or embedded programming) as well as some of the many exciting tools and projects using Ada.

Ada Developer Room Presentations (room: AW1.125, 76 seats)

The presentations in the Ada DevRoom start after the opening FOSDEM keynote. The program runs from 10:30 to 19:00, and consists of 7.5 hours with 9 talks by 9 presenters from 5 different countries, plus

2 half-hour sessions with informal discussions.

10:30-11:00 - Arrival & Informal Discussions

Feel free to arrive early, to start the day with some informal discussions while the set-up of the DevRoom is finished.

11:00-11:05 - Welcome
by Dirk Craeynest - Ada-Belgium

Welcome to the Ada Developer Room at FOSDEM 2018, which is organized by Ada-Belgium in cooperation with Ada-Europe. Ada-Belgium and Ada-Europe are non-profit organizations set up to promote the use of the Ada programming language and related technology, and to disseminate knowledge and experience into academia, research and industry in Belgium and Europe, resp. Ada-Europe has member-organizations, such as Ada-Belgium, in various countries, and direct members in many other countries. More information on this DevRoom is available on the Ada-Belgium web-site (see URL above).

11:05-11:50 - An Introduction to Ada for Beginning and Experienced Programmers
by Jean-Pierre Rosen - Adalog

An overview of the main features of the Ada language, with special emphasis on those features that make it especially attractive for free software development. Ada is a feature-rich language, but what really makes Ada stand-out is that the features are nicely integrated towards serving the goals of software engineering. If you prefer to spend your time on designing elegant solutions rather than on low-level debugging, if you think that software should not fail, if you like to build programs from readily available components that you can trust, you should really consider Ada!

12:00-12:50 - Making the Ada_Drivers_Library: Embedded Programming with Ada
by Fabien Chouteau - AdaCore

The Ada programming language was designed for embedded programming and it is well known in the aerospace domains and in general every domain where failure is not an option. Unfortunately it is not used a lot in the embedded FOSS community. In the past two years, AdaCore worked to promote the use of Ada in the FOSS community, in particular for embedded programming with the "Make with Ada" blog post series, my interview for the Embedded.fm podcase, blog posts on "ARM Community" or the "Make with Ada" competition.

In this 45 minutes lecture I will:

- give a short introduction of Ada for embedded and how its features (programming by contract, strong typing, representation clauses (hardware mapping), OOP, static compiler checks and optional run-time checks) can help improving the development time, maintenance and quality of FOSS embedded projects;
- present the Ada_Drivers_Library project, where we put all those features in practice to develop micro-controller device drivers in Ada;
- make a quick getting started demo;
- present some of the best projects from the "Make with Ada" competition.

13:00-13:20 - Shared Memory Parallelism in Ada: Load Balancing by Work Stealing
by Jan Verschelde - University of Illinois at Chicago

Tasking in Ada provides an effective tool for shared memory parallelism. For coarse grained regular parallelism, load balancing works with one single job queue. For finer grained and irregular parallelism, work stealing balances the load with multiple job queues. The programming concepts will be illustrated with examples of algorithms in polyhedral geometry. The demonstrated code belongs to the free and open source PHCpack.

13:30-13:50 - Ada, or How to Enforce Safety Rules at Compile Time
by Jean-Pierre Rosen - Adalog

This is a real life story of a mixed criticality system, where a proper usage of Ada's features for controlling visibility allowed a provable enforcement of the segregation rules at compile time: any violation would simply not compile.

14:00-14:50 - Contract-based Programming: a Route to Finding Bugs Earlier
by Jacob Sparre Andersen - JSA Research & Innovation

Contract-based programming is a software development technique, where you include assertions of program properties as a part of the compiled source text. In the strict form, the assertions are checked at compile-time, but in this presentation I will focus on the more common, less strict, form, where at least some of the assertions aren't checked until run-time. Ada gives us a lot of help, so we can write the our assertions about the program properties once, and then have the compiler insert actual run-time checks wherever there is a possibility that the assertion is violated.

This presentation will focus on how we can write these contracts in Ada in a way that make them effective at ensuring that our source text does what we intend it to and allow the compiler to generate efficient checks of the assertions. The intended audience is anybody with enough programming experience to know concepts like types, encapsulation and packages. Having tried to write Ada before will be a benefit, but it isn't a requirement.

15:00-15:50 - SPARK Language: Historical Perspective & FOSS Development
by Yannick Moy - AdaCore

SPARK started in 1987 as a restricted subset of Ada 83, defined by its own grammar rules. The overhaul of the language and toolset starting in 2010 increased greatly the language subset, dropping in effect the need for separate grammar rules. Since then, SPARK has progressively adopted most of the Ada features, to a point where the last remaining non-SPARK significant Ada feature today is pointers. We have started work on including safe pointers in SPARK, borrowing the ideas of pointer ownership from Rust. So one can legitimately wonder what difference remains between SPARK and Ada.

In the first part of this talk, I will lay out the principles that have guided us through the inclusion of language features in SPARK since 2010. I will describe in particular the trade-offs that we considered for support of important features like recursion, types with non-static constraints, generics, object orientation, concurrency. I will give a preview of the support envisioned for pointers in SPARK. So that the distinction between Ada and SPARK appears clearly: it's not about quantity, it's about safety and security.

In the second part of this talk, I will give a tour of FOSS projects which are using SPARK today: Aida, Certyflie, Muen, PolyORB-HI, Pulsar, StratoX, Tokeneer. For each one, I will describe at which level of assurance SPARK is used, with how much efforts and for which benefits. Then I will focus on the largest one, Muen, an x86/64 separation kernel for high assurance. Finally, we will look at the resources which are available to the community for FOSS development in SPARK.

16:00-16:50 - Writing REST APIs with OpenAPI and Swagger Ada
by Stephane Carrez - Bouygues Telecom

The OpenAPI specification is an emerging specification to describe RESTful web services. The Swagger suite is a collection of tools to write such API descriptions and have the code generated in more than 29 languages, including Ada. The presentation will describe how to write a REST operation with OpenAPI, generate the Ada client with Swagger Codegen and use the generated code to interact with the server. We will also describe the generated Ada server code and how to implement the server side and run a complete REST server.

17:00-17:50 - Browser-as-GUI and Web Applications with Gnoga
by Jeffrey R. Carter - Atos Belgium

Gnoga is an all-Ada library that uses the features of modern web browsers as a portable GUI. The program may run on the same computer as the browser, or on a server over the internet. Participants will be introduced to using Gnoga to create such programs.

A singleton version of the Random_Int demo program will be used to demonstrate the use of Gnoga as the GUI for a program running on the same computer as the browser. Random_Int is a very simple program that generates (pseudo)random integers in a user-specified range. The Chattanooga demo is a text-chat server program allowing people to chat on line. It demonstrates the use of Gnoga to create web applications. A secure version of Chattanooga can sometimes be accessed at https://chat.gnoga.com/. (The certificate for this site has expired, but can still be used to ensure encrypted communication with the site.)

After installing Gnoga, the demos are available in the demo directory. More information about Gnoga may be found at gnoga.com, especially the Tools page.

18:00-18:20 - Easy Ada Tooling with Libadalang
by Raphaël Amiard and Pierre-Marie De Rodat - AdaCore

A lot of developers consider that a language is only as good as the tooling that accompanies it. Ada has been conceived as a language pretty well amenable to tooling, yet the tooling offer besides AdaCore's is not very extensive, at least when compared to other languages like Java, despite the existence of the ASIS project (Ada Semantic Interface Specification).

One of Libadalang's aims is to help solve that by providing an easy way to build new Ada-aware tools. Libadalang is a library that allows the user to query information about Ada code, including:

- Syntactic information. Query the token stream, the syntax tree, find syntax patterns, etc.
- Semantic information, such as which declaration an identifier references, the type of expressions, all references to a declaration, etc.

In addition, one of the aims is to allow the users to modify the trees, and propagate the changes to the source.

This talk will go over what Libadalang can already do today, how it differs from ASIS, future plans for the library, and potential exciting use cases.

18:30-19:00 - Informal Discussions & Closing

Informal discussion on ideas and proposals for future events.

More information on Ada DevRoom

Speakers bios, pointers to relevant information, links to the FOSDEM site, etc., are available on the Ada-Belgium site at
<http://www.cs.kuleuven.be/~dirk/ada-belgium/events/18/180203-fosdem.html>

We invite you to attend some or all of the presentations: they will be given in English. Everybody interested can attend FOSDEM 2018; no registration is necessary.

We hope to see many of you there!

From: Leff Ivanov <droiddermo@gmail.com>
Date: Wed, 7 Feb 2018 00:26:35 -0800
Subject: FOSDEM 2018 Ada!
Newsgroups: comp.lang.ada

It seems this year awesome people from FOSDEM finally made things right and videos from Ada DevRoom is available for us to watch:
https://fosdem.org/2018/schedule/track/ada/

[...]

[See also “FOSDEM 2018”, AUJ 38-4, p. 175. —sparre]

Ada-Europe 2018 in Lisbon

From: Dirk Craeynest <dirk@cs.kuleuven.be>
Date: Thu, 1 Feb 2018 21:22:36 -0000
Subject: FINAL CfP Ada-Europe 2018, Mon 5 February submission deadline
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

FINAL Call for Papers

23rd International Conference on Reliable Software Technologies - Ada-Europe 2018

18-22 June 2018, Lisbon, Portugal

http://www.ada-europe.org/conference2018

Organized by Univ. Lisboa and Ada-Europe, in cooperation with ACM SIGAda, SIGBED, SIGPLAN and the Ada Resource Association (ARA)

The 23rd International Conference on Reliable Software Technologies - Ada-Europe 2018 will take place in Lisbon, Portugal. Following its traditional style, the conference will span a full week, including a three-day technical program and vendor exhibition from Tuesday to Thursday, along with parallel tutorials and workshops on Monday and Friday. This edition features a focused Special Session on Security in Safety-Critical Systems.

DEADLINE Monday 5 FEBRUARY 2018

Regular & Special Session Papers + Industrial Presentations: submit via https://easychair.org/conferences/?conf=adaeurope2018

Tutorials & Workshops: submit to the Tutorial & Workshop Chair

David Pereira <dmrpe at isep.ipp.pt>

For more information please see the full Call for Contributions at http://www.ada-europe.org/conference2018

[See also “Ada-Europe 2018 in Lisbon”, AUJ 38-4, p. 175. —sparre]

IRTAW 2018

From: Jorge Real <jorge@disca.upv.es>
Date: Sat, 10 Feb 2018 02:30:22 -0800
Subject: IRTAW 2018 Call for Papers Deadline Extension and Final Reminder
Newsgroups: comp.lang.ada

The paper submission deadline for the 19th International Real-Time Ada Workshop, IRTAW 2018 has been extended to February 14, 2018.

19th IRTAW - Call for Papers Deadline Extension and Final Reminder

The 19th International Real-Time Ada Workshop will be held on 18-20 April 2018 at Hotel Voramar, Benicassim, Spain.

The call for papers is posted at
<http://www.ada-europe.org/irtaw2018/IRTAW_2018.html>
and closes 14 February 2018.

The workshop series is famous for creating the Ravenscar Tasking Profile for Ada, plus many improvements in the area of real-time programming to every revision of Ada since Ada 95. Since 16th IRTAW, the workshop has been considering multicore paradigms for real-time systems.

If you have any interest in the areas that the workshop covers, you are invited to submit a position paper to the workshop.

Ada Semantic Interface Specification (ASIS)

ASIS/libadalang

From: Mark Lorenzen <mark.lorenzen@gmail.com>
Date: Fri, 22 Dec 2017 00:28:21 -0800
Subject: Re: ASIS for gnat GPL 2017
Newsgroups: comp.lang.ada

> [...]

I think AdaCore is phasing out support for ASIS as they migrate to libadalang instead, but I'm not absolutely sure.

https://github.com/AdaCore/libadalang

https://github.com/AdaCore/libadalang-tools

http://blog.adacore.com/cross-referencing-ada-with-libadalang

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Fri, 22 Dec 2017 10:12:53 +0100
Subject: Re: ASIS for gnat GPL 2017
Newsgroups: comp.lang.ada

> [...]

It is the official line of the party, however I think they won't do that until libadalang gets sufficiently stable to allow all their current ASIS tools to migrate - and my gut feeling is that it won't happen shortly. I've also heard rumors that they would build an ASIS layer on top of libadalang - quite sensible to maintain compatibility.

Ada-related Resources

OpenDO Forge Shutting Down

From: Open-Do Forge
Date: Wed, 31 Jan 2018 23:55:10 +0100
Subject: Open-DO Forge Shutting Down on Feb 14th

If you receive this email, then that means you are a member of one of the projects hosted on AdaCore's Open-DO forge (https://forge.open-do.org/softwaremap/full_list.php). As previously announced to all Project Admins, we are shutting down the forge and will stop hosting projects.

As of February 14th (in 2 weeks), the forge will be officially closed. Please take this time to review the projects and archive anything you need. Several of the projects have been moved to places like GitHub; this information is reflected in the project descriptions.

Do not hesitate to send an email to forge@adacore.com if you have any questions.

Ada on Social Media

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Tue Mar 6 2018
Subject: Ada on Social Media

Ada groups on various social media:

- LinkedIn:	2_698 members	[1]
- Reddit:	1_712 readers	[2]
- StackOverflow:	964 followers	[3]
- Google+:	754 members	[4]
- Freenode	87 participants	[5]
- Gitter:	53 people	[6]
- Twitter:	21 tweeters	[7]

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

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

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

[4] https://plus.google.com/communities/102688015980369378804

[5] #Ada on irc.freenode.net

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

[7] https://twitter.com/search?f=realtime&q=%23AdaProgramming

[See also "Ada on Social Media", AUJ 38-4, p. 175. —sparre]

Repositories of Open Source Software

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Tue Mar 6 2018
Subject: Repositories of Open Source software

GitHub:	2_155 repositories	[1]
	493 developers	[2]
	2_390 issues	[3]
Rosetta Code:	645 examples	[4]
	33 developers	[5]
	0 issues	[6]
Sourceforge:	233 projects	[7]
BlackDuck OpenHUB:	177 projects	[8]
Bitbucket:	94 repositories	[9]
Codelabs:	45 repositories	[10]
OpenDO Forge:	24 projects	[11]
	551 developers	[11]
AdaForge:	8 repositories	[12]

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

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

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

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

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

[6] http://rosettacode.org/wiki/Category:Ada_examples_needing_attention

[7] http://sourceforge.net/directory/language%3Aada/

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

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

[10] http://git.codelabs.ch/

[11] https://forge.open-do.org/

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

[See also "Repositories of Open Source Software", AUJ 38-4, p. 175. —sparre]

Ada-related Tools

GNAT Community Edition

From: Emma Adby <adby@adacore.com>
Date: Thu, 2 Nov 2017 13:08:09 +0000
Subject: [AdaCore] New GNAT Community edition!
To: libre-news@lists.adacore.com

Dear GNAT Community,

We are pleased to announce that AdaCore has a brand new website complete with its own section dedicated to the community. This now houses the GNAT Community edition download, formerly known as the GNAT GPL, along with other information and resources to help you get started with Ada and SPARK.

Therefore, the Libre site has gone and will now redirect you here: <https://www.adacore.com/community>, where you will find everything you need.

We hope you enjoy our new and improved community resources!

[See also "GNAT GPL and SPARK GPL", AUJ 37-2, p. 75. —sparre]

OS Command Execution

From: Victor Porton <porton@narod.ru>
Date: Mon, 13 Nov 2017 18:38:48 +0200
Subject: Execute an OS command and capture output
Newsgroups: comp.lang.ada

I need a (preferably portable, but only Linux would be well) way to execute an OS command with some input in stdin which I provide and capture the output from stdout.

Note that stdin and stdout may possibly pass NUL characters.

I am interested in both shell commands for different shells and in direct (such as execve()) execution of a command.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 13 Nov 2017 18:41:56 +0100
Subject: Re: Execute an OS command and capture output
Newsgroups: comp.lang.ada

[...]

Anyway see:

1. GNAT's System.OS_Lib.Spawn, System.OS_Lib.Non_Blocking_Spawn.

2. A more comfortable method would be to use GTK bindings (GLib actually):

http://www.dmitry-kazakov.de/ada/gtkada_contributions.htm#10

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 15 Nov 2017 19:00:49 -0600
Subject: Re: Execute an OS command and capture output
Newsgroups: comp.lang.ada

> [...]

There is no portable solution, as Ada does not provide any sort of OS access. You'll have to use something either compiler-specific (like the GNAT Spawn mentioned elsewhere or the Janus/Ada Prog_Call) or perhaps something in a library (most of which only work with GNAT anyway).

We (the ARG) once tried to define a portable OS access. We discussed it for the better part of an hour and ended up arguing about a portable problem statement -- we couldn't even agree on how to portably describe the problem we were trying to solve. We eventually decided that we could make better use of our time working on something else (pretty much anything else!). So it's unlikely that Ada will ever get such a facility; it is very target-specific. It would be nice if all Windows compilers supported the same mechanism, but that would require herding cats (implementers) and probably would be a long shot. [Implementers don't even seem to support file information portably, even though the specification of Ada.Directories.Information for Windows and for Linux is provided in the AARM. For instance, my understanding is that GNAT doesn't implement either of these packages as specified there.]

From: Andrea Cervetti <andrea.cervetti@gmail.com>
Date: Tue, 14 Nov 2017 02:57:26 -0800
Subject: Re: Execute an OS command and capture output
Newsgroups: comp.lang.ada

> [...]

See the example http://rosettacode.org/wiki/Get_system_command_output#Ada

It uses GNAT.Expect.Get_Command_Output that does exactly what you need.

You have just to define a String for the Input parameter to the Get_Command_Output function.

Qt5Ada

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Sat, 9 Dec 2017 03:02:27 -0800
Subject: Announce : Qt5Ada version 5.10.0 (546 packages) release 09/12/2017 free edition
Newsgroups: comp.lang.ada

Qt5Ada is Ada-2012 port to Qt5 framework (based on Qt 5.10.0 final)

Qt5ada version 5.10.0 open source and qt5c.dll,libqt5c.so(x64) built with Microsoft Visual Studio 2015 in Windows, gcc x86-64 in Linux.

Package tested with gnat gpl 2012 ada compiler in Windows 32bit and 64bit , Linux x86-64 Debian 8.5

It supports GUI, SQL, Multimedia, Web, Network, Touch devices, Sensors,Bluetooth, Navigation and many others thinks.

Changes for new Qt5Ada release :

Added direct load UI QDesigner generated files:

.New packages and demos

My configuration script to build Qt 5.10.0 is: configure -opensource -release -nomake tests -opengl dynamic -qt-zlib -qt-libpng -qt-libjpeg -openssl-linked OPENSSL_LIBS="-lssleay32 -llibeay32" -plugin-sql-mysql -plugin-sql-odbc -plugin-sql-oci -icu -prefix "e:/Qt/5.10"

As a role ADA is used in embedded systems, but with QTADA(+VTKADA) you can build any desktop applications with

powerful 2D/3D rendering and imaging (games, animations, emulations) GUI, Database connection, server/client, Internet browsing , Modbus control and many others thinks.

Qt5Ada and VTKAda for Windows, Linux (Unix) is available from

https://drive.google.com/folderview?id=0B2QuZLoe-yiPbmNQRl83M1dTRVE&usp=sharing

(google drive. It can be mounted as virtual drive or directory or viewed with Web Browser)

The full list of released classes is in "Qt5 classes to Qt5Ada packages relation table.docx"

VTKAda version 8.0.0 is based on VTK 8.0.0 (OpenGL2) is fully compatible with Qt5Ada 5.10.0

I hope Qt5Ada and VTKAda will be useful for students, engineers, scientists and enthusiasts

With Qt5Ada you can build any applications and solve any problems easy and quickly.

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

[See also “Qt5Ada”, AUJ 38-2, p. 74. —sparre]

AdaControl

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Fri, 15 Dec 2017 17:02:29 +0100
Subject: [Ann] New version of AdaControl (with terrific new features)
Newsgroups: comp.lang.ada

Adalog is pleased to announce the release of version 1.19r10 of AdaControl, with a big count of 539 rules and subrules.

Special mention to the new subrule Assignment/Access_Duplication: it controls assignments where at least one subcomponent is of an access type. Great for finding remaining references after an Unchecked_Deallocation.

BUT THERE IS MORE TO IT:

AdaControl can now generate suggested fixes in its output file. Under GPS, the corresponding messages are marked with the small "wrench" icon, and clicking on it applies the fix. There is also an adactl_fix utility that applies automatically all suggested fixes.

Remember these hundreds of "wrong casing" violations that you never had the time to fix? It now takes seconds to reduce your technical debt!

AND MORE:

You can now download plugins that provide full integration of AdaControl with GnatHub/GnatDashboard/SONARQube!

Full list of improvements is available in file HISTORY as usual.

AdaControl can be downloaded from http://www.adacontrol.fr

Enjoy!

[See also “AdaControl”, AUJ 37-4, p. 190. —sparre]

GStreamer Binding

From: Per Sandberg <per.s.sandberg@bahnhof.se>
Date: Mon, 25 Dec 2017 21:17:57 +0100
Subject: [ANN] version alpha 0.0 of Ada GStreamer bindings.
Newsgroups: comp.lang.ada

Just want to take a poll on the interest level.

Got Ada-equivalents for all GStreamer header files.

https://github.com/persan/A-gst

ArchiCheck

From: Lionel Draghi <lionel.draghi@gmail.com>
Date: Sat, 30 Dec 2017 14:01:40 -0800
Subject: [ANN] Archicheck v 0.5.0
Newsgroups: comp.lang.ada

A new ArchiCheck version is available.

Most important changes since v0.3:

1. Many improvements in the rules syntax : cf.

   http://lionel.draghi.free.fr/Archicheck/rules/

2. A first implementation of Java support

Quick Start : http://lionel.draghi.free.fr/Archicheck/

Feel free to suggest any Ada or Java software, with a minimally described architecture (like this : https://xmlgraphics.apache.org/batik/using/architecture.html), and available sources, so that I can improve my test suite.

[See also “ArchiCheck”, AUJ 38-2, p. 74. —sparre]

GtkAda Contributions

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 4 Jan 2018 10:02:12 +0100
Subject: ANN: GtkAda contributions 3.19
Newsgroups: comp.lang.ada

The library extends GtkAda bindings to GTK+. It deals with the

following issues:

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

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

The new version adds SVG images support through bindings to librsvg2.

[See also “GtkAda Contributions”, AUJ 38-1, p. 5. —sparre]

Industrial Control Widget Library

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 5 Jan 2018 10:12:31 +0100
Subject: ANN: Ada industrial control widget library v 3.17
Newsgroups: comp.lang.ada

The library assists design of high-quality industrial control widgets for Ada applications. The software is based on GtkAda, Ada bindings to GTK+ and cairo. The key features of the library:

- Widgets composed of transparent layers drawn by cairo;
- Fully scalable graphics;
- Support of time controlled refresh policy for real-time and heavy-duty applications;
- Caching graphical operations;
- Stream I/O support for serialization and deserialization;
- Ready-to-use gauge, meter, oscilloscope widgets;
- Editor widget for WYSIWYG design of complex dashboards.

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

Changes to the previous version:

- The package Gtk.Layered.Line has color opacity parameter added;
- The package Gtk.Layered.SVG layer was added to show an SVG image;
- The package Gkt.Layered.Disk_Needle was added, a needle type used in valve position indicating instruments;
- The package Gtk.Valve.Round_90 example added, it provides a valve position indicator;
- Bordered layers have "lens" with light reflex and shadow;
- Another bug fix in Gtk.Layered.Editor to work around GNAT compiler issues;
- Minor bug fixes.

[See also “Industrial Control Widget Library”, AUJ 38-4, p. 177. —sparre]

VTKAda

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Thu, 4 Jan 2018 23:39:13 -0800
Subject: VTKAda version 8.1.0 release 05/01/2018
Newsgroups: comp.lang.ada

I'm pleased to announce VTKAda version 8.1.0 free edition release 05/01/2018. VTKAda is Ada-2012 port to VTK (Visualization Toolkit by Kitware, Inc) and Qt5 application and UI framework by Nokia.

VTK version 8.1.0, Qt version 5.10.0 open source and vtkc.dll, vtkc2.dll, qt5c.dll(libvtkc.so,libvtkc2.so,libqt5c.so) were built with Microsoft Visual Studio 2015 in Windows (WIN32) and gcc in Linux x86-64 Package was tested with gnat gpl 2017 ada compiler in Windows 0 64bit,Debian 9.2 x86-64.

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

VTKADA you can be used without QTADA subsystem Qt5Ada and VTKAda for Windows, Linux (Unix) is available from https://drive.google.com/folderview?id=0B2QuZLoe-yiPbmNQRl83M1dTRVE&usp=sharing (google drive. It can be mounted as virtual drive or directory or viewed with Web Browser)

[See also “VTKAda”, AUJ 38-3, p. 117. —sparre]

GNAT: Stack Traces

From: Anh Vo <anhvofrcaus@gmail.com>
Date: Mon, 15 Jan 2018 09:22:34 -0800
Subject: Re: ADA.STRINGS.INDEX_ERROR : a-strunb.adb:782
Newsgroups: comp.lang.ada

> [...]

It is even better to use GNAT addition to print out full stack trace as shown in the snippet.

```
exception
   when Err : others =>
      Text_IO.Put_Line ("Houston we have a
            problem: " &
            Exceptions.Exception_Information
                     (Err));
      Text_IO.Put_Line ("Traceback => " &
            GNAT.Traceback.
            Symbolic.Symbolic_Traceback
                     (Err));
end [Ada Unit Name];
```

[See also “Traceback Wrapper”, AUJ 37-1, p. 7. —sparre]

Raspberry Pi SenseHAT

From: Philip Munts <philip.munts@gmail.com>
Date: Thu, 25 Jan 2018 03:12:56 -0800
Subject: Re: Raspberry Pi SenseHAT / AstroPi
Newsgroups: comp.lang.ada

I have written some Ada packages and test programs for the Sense HAT. The platform independent sensor drivers are in libsimpleio:

http://git.munts.com/libsimpleio/ada/

The platform dependent code is in MuntsOS:

http://git.munts.com/arm-linux-mcu/examples/ada

My code use the i2c-dev interface rather than kernel drivers. I had to reverse engineer the display and joystick from the AVR firmware source code.

I primarily target my own embedded Linux distribution, MuntsOS, but the test programs all run on Raspbian as well.

List_Image

From: Lionel Draghi <lionel.draghi@gmail.com>
Date: Tue, 30 Jan 2018 16:44:07 -0800
Subject: [ANN] List_Image v0.2.0
Newsgroups: comp.lang.ada

List_Image is a small helper to print the content of Ada predefined containers, available here: https://github.com/LionelDraghi/List_Image

The Image generic function returns the image of containers content in various format, customizable at instantiation time, from the simple

```
A, B, C
```

or

```
[A, B, C]
```

or

```
- A
- B
- C
```

or

```
A, B and C
```

or more complex format like html :

```
<ul>
<li>A</li>
<li>B</li>
<li>C</li>
</ul>
```

Format may differ for empty list and list containing a single item.

So, the same instantiation of the function may return :

- "No test failed" if the list is empty
- "Test test_1 fails" if the list contains "test_1"
- "Tests test_1 and test_2 fail" if the list contains "test_1" and "test_2"

ID3 Parser

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Wed, 7 Feb 2018 21:23:37 -0800
Subject: any mp3 library bindings?
Newsgroups: comp.lang.ada

I'm working on a web interface to my music library, and I need to read the meta info from each music file; i.e., read the mp3 tags.

I found ffmpeg (https://ffmpeg.org/), which provides a C interface. Does anyone have Ada bindings for that, or another mp3 library?

From: Per Sandberg <per.s.sandberg@bahnhof.se>
Date: Thu, 8 Feb 2018 17:49:42 +0100
Subject: Re: any mp3 library bindings?
Newsgroups: comp.lang.ada

Well gave it some 30 minutes and ended up with a 1:1 mapping of between Ada-specs and C-headers:

https://github.com/persan/a-ffmpeg

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Fri, 9 Feb 2018 15:24:58 -0800
Subject: Re: any mp3 library bindings?
Newsgroups: comp.lang.ada

> [...]

> https://github.com/persan/a-ffmpeg

I tried writing my own subset of that, since I only need a few subprograms. It linked against the mingw32 ffmpeg libraries, but crashed at runtime, before the GNAT debugger got control.

So I wrote my own ID3 parser (http://id3.org/d3v2.3.0); much simpler, and it works!

From: Per Sandberg <per.s.sandberg@bahnhof.se>
Date: Sat, 24 Feb 2018 15:52:19 +0100
Subject: Re: any mp3 library bindings?
Newsgroups: comp.lang.ada

> [...]

Well I had to generate a binding to "http://taglib.org/". The API is Ada-style. See:

https://github.com/persan/a-taglib

ANSI Terminal Control (X3.64)

From: Edward R. Fish <onewingedshark@gmail.com>
Date: Fri, 16 Feb 2018 15:47:52 -0800
Subject: Ada package for handling ANSI Standard (X3.64)
Newsgroups: comp.lang.ada

I've looked around and haven't been able to find any X3.64 library/bindings for Ada, does anyone know of any? -- I *think* ISO-2022 library/bindings might work, but I'd have to double-check on that.

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Sat, 17 Feb 2018 09:54:47 +0100
Subject: Re: Ada package for handling ANSI Standard (X3.64)
Newsgroups: comp.lang.ada

> [...]

There's a partial implementation in PragmARC.ANSI_TTY_Control; additional sequences are easy to add.

https://github.com/jrcarter/PragmARC

From: Emmanuel Briot <briot@adacore.com>
Date: Sat, 17 Feb 2018 03:17:43 -0800
Subject: Re: Ada package for handling ANSI Standard (X3.64)
Newsgroups: comp.lang.ada

Also GNATCOLL.Terminal will let you output colored text (including on Windows where by default the terminal does not support ANSI escape sequences).

To interpret those sequences, there is a package in GPS (widgets/src/gtkada-terminal.c) that provides a GtkAda text viewer. Perhaps it could be moved outside of GPS (it wasn't tricky to write, but now it is able to interact with vim, for instance, which is not bad)

Ada-related Products

GNAT Pro, CodePeer, SPARK Pro and QGen

From: AdaCore Press Center
Date: Wed Jan 31 2018
Subject: AdaCore V18.1 Product Release Brings New Software Development and Verification Solutions
URL: https://www.adacore.com/press/adacore-v18-1-product-release-brings-new-software-development-and-verification-solutions

GNAT Pro, CodePeer, SPARK Pro and QGen upgraded with new features, better performance

NEW YORK & PARIS & TOULOUSE, France, January 31, 2018 - ERTS² Congress - AdaCore, a trusted provider of software development and verification tools with headquarters in New York and Paris, is unveiling Version 18.1 of its industry-leading GNAT Pro, CodePeer, SPARK Pro, and QGen products. This latest version enhances the already robust, feature-rich software suite with new tools and capabilities and marks the first release of GNAT Pro in its three product lines: GNAT Pro Assurance, GNAT Pro Enterprise and GNAT Pro Developer.

For more than twenty years, AdaCore has been delivering products and services designed specifically to help developers build safe, secure, reliable, high-integrity software. AdaCore's GNAT Pro, CodePeer, SPARK Pro and QGen are software development and verification tools of choice for safety-critical, high-security, and mission-critical applications across a growing number of markets, including aerospace and defense, automotive, energy, medical, railway, and the Internet of Things (IoT).

“Specializing the GNAT Pro offering into three product lines is one of the major new aspects of this release, allowing us to adapt our toolset and services to different user needs” said Cyrille Comar, AdaCore President. “As an example, the GNAT Pro Assurance product line provides specific services for environments where a major tooling upgrade can be very delicate and where tool errata (‘known-problems’) need to be accurately described and actively managed.”

The GNAT Pro Ada development environment provides new tools, improved code efficiency, several new targets and a variety of compiler enhancements with Version 18.1. All GNAT Pro subscriptions now include the SPARK Discovery verification technology, and GNAT Pro Assurance and GNAT Pro Enterprise also supply AdaCore's GNATstack stack analysis tool.

The GNAT Pro tool suite features GPRbuild project tool upgrades, new rules in the GNATcheck coding standard checker, GNATcoverage support for Lauterbach probes, and new options in the GNATtest unit testing framework. The GNAT Programming Studio (GPS) Integrated Development Environment (IDE) boasts performance and user interface improvements, including C/C++ navigation engine enhancements, while GNATbench adds support for Eclipse 4.8 Oxygen and Wind River Workbench 4.12.

CodePeer, the advanced CWE-compatible static analysis tool for Ada, brings improved performance, easier analysis of non-GNAT code, fewer “false positives”, and integration of GNAT warnings into the tool output. An updated tool qualification kit for DO-178C is also available for CodePeer 18.1, with coverage of additional Ada constructs including exceptions, access types, and generic units.

The SPARK Pro formal verification environment, co-developed by AdaCore and its partner Altran, has added contracts to units in the predefined environment and has enhanced the automation of proofs. SPARK Pro 18.1 also allows users to perform interactive proofs in GPS.

QGen, AdaCore’s qualifiable model-based development tool kit which generates SPARK and MISRA C, has been enhanced with a compatibility checker to verify compliance of sample models with the supported safe subset of Simulink®/Stateflow® models. QGen 18.1 also implements several new blocks and supports Simulink®/Stateflow® versions up to 2017b.

For a complete list of new and improved features in Version 18.1, visit:

- GNAT Pro base technology – http://docs.adacore.com/R/relnotes/features-18
- GPS and GNATbench IDEs – http://docs.adacore.com/R/relnotes/features-ide-18
- GPR library and tools – http://docs.adacore.com/R/relnotes/features-gprbuild-18 CodePeer – http://docs.adacore.com/R/relnotes/features-codepeer-18
- SPARK Pro – http://docs.adacore.com/R/relnotes/features-spark-18
- QGen – http://docs.adacore.com/R/relnotes/features-qgen-18

[...]

Ada and CORBA

OMG IDL-4.x Mapping Revision?

From: Oliver Kellogg <olivermkellogg@gmail.com>
Date: Tue, 2 Jan 2018 12:05:55 -0800
Subject: OMG IDL-4.x to Ada mapping revision
Newsgroups: comp.lang.ada

Thanks to the IDL4 "building blocks" [1], we have clean separation of the common IDL core from transport specific aspects (CORBA, CCM, DDS, etc.)

The IDL to C++11 mapping [2] has taken steps toward IDL4. The current IDL to Ada mapping [3] is still closely tied in with CORBA. Is anyone working on aligning it with IDL4?

[1] http://www.omg.org/spec/IDL/4.2/Beta1/

[2] http://www.omg.org/spec/CPP11/1.3/Beta1/

[3] http://www.omg.org/spec/ADA/1.3/

Ada and Operating Systems

Mac OS X: ASIS for GNAT GPL

From: Rasika Srinivasan <s@srin.me>
Date: Thu, 21 Dec 2017 20:15:37 -0800
Subject: ASIS for gnat GPL 2017
Newsgroups: comp.lang.ada

My gnat 2017 installs on macos does not appear to include ASIS. Nor have I been able to download from the Adacore site.

I am hoping to find a solution that is usable in macos and linux. pointers appreciated.

From: Simon Wright <simon@pushface.org>
Date: Fri, 22 Dec 2017 10:29:39 +0000
Subject: Re: ASIS for gnat GPL 2017
Newsgroups: comp.lang.ada

> [...]

On the Libre (Community Edition) download page for Mac OS X (btw, should be macOS now!!), at the bottom right, click on "More packages, platforms, versions and sources"; click on the "Sources" link under "GNAT GPL Ada"; asis-gpl-2017-src.tar.gz is the second source listed.

Mac OS X: JVM-GNAT-GPL

From: Pascal Pignard <p.p14@orange.fr>
Date: Sun, 4 Mar 2018 20:53:35 +0100
Subject: JVM-GNAT GPL 2013 binaries for macOS available
Newsgroups: gmane.comp.lang.ada.macosx

I've upload JVM-GNAT GPL 2013 binaries for macOS El Capitan on Source Forge:

https://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2017-el-capitan/

This is the Ada GNAT compiler targeted for the Java Virtual Machine.

Some complete examples are on:

https://github.com/Blady-Com/jvm-gnat-examples.

There are also some bugs reported there.

Well, GNAT for JVM seems to have been no more supported by AdaCore since 2013. But recently I received some motivation from Ivan Levashev which had succeeded running an applet with GNAT-JVM Windows version:

https://plus.google.com/+ИванЛевашев/posts/i7cHtH7chMo

Furthermore, he succeeded in bringing JRE classes translated in Ada:

https://gitlab.ow2.org/octagram/Hello_CheerpJGNAT

I hope this may be useful. Nevertheless, don't hesitate to report here if you try it.

[See also “JVM-GNAT GPL 2011 for Mac OS X Snow Leopard”, AUJ 33-1. —sparre]

References to Publications

Using Ada.Sequential_IO to Create a Simple Hexdump Utility

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Mon Dec 26 2016
Subject: Using Ada.Sequential_IO to create simple hexdump utility
URL: http://ada.tips/using-adasequential_io-to-create-simple-hexdump-utility.html

There is every now and then need for viewing the file contents as "raw" bytes shown as hex numbers. Usually, operating systems have tool 'hexdump' to do this. But it is not hard to create one from scratch.

Here is one example how to create a simple hexdump utility in Ada.

[...]

On Sponsoring Open Source Software

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Sun, 12 Nov 2017 06:21:15 +0100
Subject: Re: Looking for experience on XML parser (DOM) for Ada
Newsgroups: comp.lang.ada

> [...]

Not exactly your question, but you can find my paper "On the benefits for industrials of sponsoring free software development" interesting: http://www.adalog.fr/publicat/Free-software.pdf

AdaTutor

From: Joakim Strandberg <joakimds@kth.se>
Date: Sat, 23 Dec 2017 12:30:16 -0800
Subject: Re: exercices
Newsgroups: comp.lang.ada

One resource for learning Ada that I like is: https://zhu-qy.blogspot.se/2012/08/adatutor.html

There are exercises there if I remember correctly.

[See also “AdaTutor on the web”, AUJ 33-4, p. 235. —sparre]

Going all-in with Ada

From: Dirk Craeynest <dirk@cs.kuleuven.be>
Date: Tue, 9 Jan 2018 11:57:29 -0000
Subject: Check out "Going all-in with Ada"
Newsgroups: comp.lang.ada
https://www.reddit.com/r/ada/comments/7p12n3/going_allin_with_ada_a_manifesto/

Another great article to read and distribute.

Thanks Ingo for pointing it out!

AVR-Ada Tutorial

From: Jordan Lee Mauro-Buhagiar <jordanleemauro@gmail.com>
Date: Fri, 26 Jan 2018 19:03:31 +0100
Subject: AVR-ADA Tutorial Now Available
Newsgroups: gmane.comp.hardware.avr.ada

First Book explaining the process

- Kindle: https://www.amazon.co.uk/Real-Time-Critical-Systems-Prototype-Integration-ebook/dp/B07986YGNM/ref=sr_1_2?ie=UTF8&qid=1516989731&sr=8-2&keywords=real+time+critical+systems
- Paperback: https://www.amazon.co.uk/dp/1984171992/ref=sr_1_1?ie=UTF8&qid=1516989731&sr=8-1&keywords=real+time+critical+systems

Ada Inside

MAX! Home Automation

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 17 Dec 2017 16:02:02 +0100
Subject: ANN: MAX! home automation v2.0
Newsgroups: comp.lang.ada

MAX! home automation is a GTK+ application to manage ELV/eQ-3 MAX! cubes. A cube is a gateway to a network of radiator thermostats, shutter contacts etc.

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

Changes to the previous version:

- MQTT broker bug fix that prevented publishing thermostat temperatures;
- MQTT broker settings allow specifying if the broker should accept publishing requests from the clients on unknown topics. The topics can be limited by a list of MQTT topic patterns.
- Python scripts added.

P.S. I would like to support scripting in an Ada-friendly interpreter in addition to Python, which I really dislike. I remember some discussions about Ada-like scripting language, but found no references to.

[See also “MAX! Home Automation”, AUJ 38-4, p. 181. —sparre]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 6 Jan 2018 10:46:30 +0100
Subject: ANN: MAX! home automation v2.3
Newsgroups: comp.lang.ada
[...]

Changes to the previous version:

- Minor bug fixed;
- Example of controlling a relay from a script.

The script is written in Python. A Wemos D1 relay is attached to a ESP8266 microcontroller. ESP8266 is connected to WiFi and controlled from the script via MQTT topic published on the MAX! home automation broker.

P.S. Proposals and help with porting to AVR Ada and/or adding scripting in an Ada-like scripts are welcome.

Artificial Heart

From: AdaCore Press Center
Date: Tue Feb 27 2018
Subject: Scandinavian Real Heart Selects AdaCore Embedded Software Development Platform for Revolutionary Artificial Heart
URL: https://www.adacore.com/press/scandinavian-real-heart-selects-adacore-embedded-software-development-platform-for-revolutionary-artificial-heart
AdaCore software development and verification tools help Real Heart deliver the high assurance, safety, and reliability that lifesaving medical devices demand.

Embedded World Booth # 4-149

NUREMBERG, Germany, February 27, 2018 – Embedded World – AdaCore, a trusted provider of software development and verification tools with headquarters in New York and Paris, today announced that Scandinavian Real Heart AB in Sweden is using a suite of AdaCore software solutions to develop reliable embedded software for its revolutionary Total Artificial Heart.

Scandinavian Real Heart’s Total Artificial Heart mimics the way that the natural heart functions to “save patients with heart failure, and give them a better quality of life than the alternatives that are available today” said Fredrik Pahlm, Chief Technology Officer (CTO) and Project Manager at Scandinavian Real Heart.

Scandinavian Real Heart is in the final development phase of the heart pump’s motor control software, which is both complicated and truly unique in its ability to adjust to the patient’s blood pressure. Real Heart employs AdaCore software solutions throughout its end-to-end embedded software development workflow, including:

- The GNAT Programming Studio (GPS) Integrated Development Environment (IDE) for designing, implementing, and managing applications that demand high reliability,
- The SPARK Pro verification toolset based on formal methods and oriented toward high-assurance systems,
- The GNAT Pro Ada for ARM multi-language development environment for use with ARM processors,
- The GPRbuild advanced build system that helps automate the construction of multi-language systems and
- The GNATstack static analysis tools for stack usage computation.

“Our heart pump has to work uninterrupted throughout the life of the patient” said Professor Lars Asplund, Main Software Architect at Scandinavian Real Heart. “The quality and reliability of all parts of the system are crucial. We want to create software with the highest level of safety, and we know that SPARK together with Ada is the best option.”

“The programming tools and programming language were selected considering optimum reliability and quality assurance” added Azad Najar, Scandinavian Real Heart Chief Executive Officer (CEO).

“Innovators like Scandinavian Real Heart continue to choose AdaCore’s comprehensive suite of software development and verification solutions, particularly for lifesaving and safety-critical applications” said Jamie Ayre, Commercial Team Lead at AdaCore. “AdaCore provides the open-source tools and libraries embedded systems developers need to craft the most complex software with high assurance, integrity, and reliability while lowering development and verification costs.”

AdaCore software solutions have been the software development and verification tools of choice for safety-critical and mission-critical applications for decades. AdaCore continues to advance and adapt its trusted tools to meet the most stringent requirements and high-assurance, high-integrity needs of modern projects across multiple markets.

AdaCore is presenting its suite of software tools – including Version 18.1 of its industry-leading GNAT Pro, CodePeer, SPARK Pro and QGen products, as well as GNAT Pro Assurance, GNAT Pro Enterprise and GNAT Pro Developer tailored to specific user needs – in Booth # 4-149 at Embedded World 2018.

The annual Embedded World exhibition and conference is taking place at the Nuremberg Exhibition Centre in Nuremberg, Germany, from 27 February through 1 March 2018.

[...]

Ada in Context

Declaring Subtypes of Private Types

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Sat Sep 2 2017
Subject: Declaring subtypes of private types
URL: http://ada.tips/declaring-subtypes-of-private-types.html

Sometimes, you want to declare a subtype for a subset of the possible values of a private type.

You might for example want a subtype of Ada.Calendar.Time, which only can include times in the past. If Ada.Calendar.Time was a numeric type, we could declare a subset using range, but as it is a private type, we have to do it differently:

```
subtype Past_Time is Ada.Calendar.Time
  with Dynamic_Predicate =>
        Past_Time < Ada.Calendar.Clock;
```

You could even expand this to a subtype representing values in the last hour:

```
subtype Last_Hour is Past_Time
  with Dynamic_Predicate =>
        Ada.Calendar.Clock - 3600.0
        <= Last_Hour;
```

Notice that while Past_Time is an ever expanding subtype, Last_Hour is an ever changing subtype, such that values which were valid earlier, not necessarily are valid now.

Adding Finalization to an Existing Type

From: Jere <jhb.chat@gmail.com>
Date: Wed, 22 Nov 2017 04:43:41 -0800
Subject: Extending a third party tagged type while adding finalization
Newsgroups: comp.lang.ada

Has anyone have any good tips for extending a third party type:

```
type Third_Party_Type is tagged private;
```

I want to extend it but also add finalization which is inheritable by later descendants:

```
type My_Base_Type is new
      Third_Party_Type with private;
procedure Finalize (Object : in out
      My_Type);
procedure Initialize (Object : in out
      My_Type);
procedure Adjust (Object : in out
      My_Type);
```

That way, clients of My_Base_Type can use them along side Third_Party_Type such as through a variable of Third_Party_Type'Class. My first (untested) thought is to maybe string together a proxy object inside My_Base_Type using the Rosen technique and have it forward the Finalization operations through dispatch. I haven't tested this to see if it works but even if it does, it feels very hacky and is not my preferred way.

Does anyone have any techniques they have used in the past?

From: Christoph Karl Walter Grein <christ-usch.grein@t-online.de>
Date: Wed, 22 Nov 2017 08:42:17 -0800
Subject: Re: Extending a third party tagged type while adding finalization
Newsgroups: comp.lang.ada

Perhaps this might help you: http://www.christ-usch-grein.homepage.t-online.de/Ada/Finalization.html

I published this in Ada Letters, Volume XIX, Number 4, December 1999.

Pre'Class

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 11 Dec 2017 16:47:26 -0600
Subject: Re: I am leaving Ada :-(because of GNAT bugs
Newsgroups: comp.lang.ada

Simon Wright wrote:

> See ARM 6.1.1(17.1/4), "Pre'Class shall not be specified for an overriding primitive subprogram of a tagged type T unless the Pre'Class aspect is specified for the corresponding primitive subprogram of some ancestor of T."

That was a rule change in the 2015 Corrigendum. It was made because the derived Pre'Class will be ored with the inherited one, and if you've inherited from a routine with no Pre'Class, you are oring with True -- which means your new Pre'Class will be completely ignored.

Pre'Class makes sense mainly for a root class; one generally will use dispatching calls within it so it can adjust as needed for child types. If you need a precondition that doesn't fit into that model, you should use Pre (and be aware that you are violating the basic LSP approach).

Empty Arrays

From: Robert I. Eachus <rieachus@comcast.net>
Date: Thu, 21 Dec 2017 21:01:30 -0800
Subject: Re: syntaxic exploration
Newsgroups: comp.lang.ada

> [...]

Is it a reasonable language improvement request to allow "" for other empty arrays? Another possibility is the reserved word null. If that were allowed I might even use it in place of "" for strings, because that is not different enough from "" when reading, depending on the type face used.

From: Simon Clubley <clubley@eisner.decus.org>
Date: Fri, 22 Dec 2017 21:15:18 -0000
Subject: Re: syntaxic exploration
Newsgroups: comp.lang.ada

> Is it a reasonable language improvement request to allow "" for other empty arrays?

No. That would be too confusing IMHO because it would initially appear to be a string type to anyone unfamiliar with that part of the code and that just feels _way_ wrong.

> Another possibility is the reserved word null. [...]

null is an interesting option for the empty array. However, for your strings idea, would people unfamiliar with the code and this new usage of null read this new usage as a null pointer instead of an empty string? That could be confusing.

The 1 .. 0 notation is ugly but anyone who sees it knows instantly exactly what the original programmer meant. Any cleaner replacement should instantly come across as the empty array without the possibility of confusion because someone reused Ada syntax for this in an ambiguous way.

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Sat, 23 Dec 2017 00:11:49 +0200
Subject: Re: syntaxic exploration
Newsgroups: comp.lang.ada

[...]

I think the aggregate form (null array) was suggested some time ago, in analogy with the existing (null record). It looks good to me, but as I remember, there was some objection. One problem is that if the index type has exactly one value, then a null array with that index type cannot exist (because then A'First = A'Last for any such array A).

> The 1..0 notation is ugly but anyone who sees it knows instantly exactly what the original programmer meant.

Yes, but it becomes rather more ugly if the index type is an enumeration, or a generic formal discrete type, something like

```
  Thingummybob'Succ (Thingummybob'First)
.. Thingummybob'First
```

Note that Thingummybob'Last .. Thingummybob'First won't be a null range if Thingummybob has exactly one value.

[...]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 22 Dec 2017 23:51:58 +0100
Subject: Re: syntaxic exploration
Newsgroups: comp.lang.ada

> I think the aggregate form (null array) was suggested some time ago, in analogy with the existing (null record).

These two totally different things. "null record" is not a record aggregate (a value of record type), it is a type construct (a value of a record type's type). Correspondingly "null array" would mean a type of arrays rather than an empty aggregate of an array type.

> I think (null array) satisfies that. I wish it could be introduced.

If "" is QK, why () isn't?

```
  Empty : My_Array := ();
```

It could be an attribute as well:

```
  Empty : My_Array := My_Array'Null;
```

(You are right about indices. String has the lower bound set to 1. That gives an unambiguous range of an empty string. For general case arrays it is ambiguous.)

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Sat, 23 Dec 2017 09:15:35 +0200
Subject: Re: syntaxic exploration
Newsgroups: comp.lang.ada

> These two totally different things. "null record" is not a record aggregate (a value of record type), it is a type construct (a value of a record type's type).

Yes and no. Compare

```
  type Nothing is null record;
  -- Last choice in RM 3.8(3).
```

and

```
  N : Nothing := (null record);
  -- Last choice in RM 4.3.1(3).
```

So "null record" in a type declaration is a record_definition (RM 3.8(3)) but it can also occur in an expression as a record_component_association_list (RM 4.3.1(3)). Thus "(null record)" is a record aggregate.

> [...]

> If "" is QK, why () isn't?

Could be, but parentheses are already so overloaded (which quotes are not) that I like (null array) better.

> [...]

> Empty : My_Array := My_Array'Null;

Could be, but it is not much better than My_Array'(null array).

> (You are right about indices. String has the lower bound set to 1. That gives an unambiguous range of an empty string.

No, for example (16 .. -40 => ' ') is an empty string, with bounds 16 .. -40.

The index range of the empty string denoted by the null string literal "" is defined in the RM unambiguously.

> For general case arrays it is ambiguous)

Yes, the index bounds of "(null array)" would have to be specifically defined in the RM, similarly to the case for the null string literal.

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Sat, 23 Dec 2017 17:23:29 +0100
Subject: Re: syntaxic exploration
Newsgroups: comp.lang.ada

> [...] One problem is that if the index type has exactly one value, then a null array with that index type cannot exist (because then A'First = A'Last for any such array A).

Actually, a null array of such a type can exist if it's a string type, because you can use the string literal "" for such types. AIUI, 'Last is undefined for such a value. Allowing (null array) for non-string array types with an index type with a single value would presumably work the same.

From: Robert I. Eachus <rieachus@comcast.net>
Date: Sat, 23 Dec 2017 19:37:40 -0800
Subject: Re: syntaxic exploration
Newsgroups: comp.lang.ada

> [...]

The rule that allows 1..0 as a null string range only has a problem if you have an array with an enumeration index type containing a single component. I think that is one of those cases where we say, "Don't do that!" and move on.

A more complex, and potentially ugly case is for multidimensional arrays. Defining (null array) as being empty in all dimensions with Foo'Range(n) = Bar'First..Bar'Pred(Bar'First) where Bar is the nth index SUBtype for Foo works. If the last materialized is a (real) Numeric_Error AKA Constraint_Error? Again, only an issue for smart alecks. In general, you should be surprised if Foo'Last(N) for a null array doesn't raise Constraint_Error.

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Sun, 24 Dec 2017 15:39:56 +0200
Subject: Re: syntaxic exploration
Newsgroups: comp.lang.ada

> [...]

> A more complex, and potentially ugly case is for multidimensional arrays. Defining (null array) as being empty in all dimensions with Foo'Range(n) = Bar'First..Bar'Pred(Bar'First) where Bar is the nth index SUBtype for Foo works.

In other words, the same as the current rule for the null string literal "".

> In general, you should be surprised if Foo'Last(N) for a null array doesn't raise Constraint_Error.

That is not true in current Ada, as I understand it.

That would be very bad -- it would mean that any general operation that loops over an array that is potentially null would have to first check the 'Length. Fortunately this is not the case.

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Sun, 24 Dec 2017 15:32:57 +0200
Subject: Re: syntaxic exploration
Newsgroups: comp.lang.ada

> [...]

RM 4.2(9) says "for a null string literal, the upper bound is the predecessor of the lower bound". I would understand this to mean applying 'Pred to the lower bound, which will raise Constraint_Error if the type has only one value.

If I try this with GNAT:

```
  type One_T is (Unique);
  type Str_T is array (One_T range <>) of
Character;
  S : constant Str_T := "";
```

I get compilation errors:

```
  nullstr.adb:11:26: null string literal not
allowed for type
     "Str_T"  defined at line 9
  nullstr.adb:11:26: static expression fails
Constraint_Check
```

where line 11 is the one with the "" literal.

So one can try to make a null array of this type with "", but it will fail.

> Allowing (null array) for non-string array types with an index type with a single value would presumably work the same.

I agree, it should also raise Constraint_Error.

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Mon, 25 Dec 2017 14:40:18 +0100
Subject: Re: syntaxic exploration
Newsgroups: comp.lang.ada

> nullstr.adb:11:26: null string literal not allowed for type

> "Str_T" defined at line 9

> nullstr.adb:11:26: static expression fails Constraint_Check

Interesting. I'm pretty sure that used to work. Has that changed in a recent revision?

Ranges over Containers

From: Brad Moore <bmoore.ada@gmail.com>
Date: Thu, 21 Dec 2017 15:46:54 -0800
Subject: Re: syntaxic exploration
Newsgroups: comp.lang.ada

> [...]

I've been thinking lately that it's a bit of a wart that 'Range or at least range syntax can´t be used in more places.

Specifically, I have been thinking of container iteration.

It is currently awkward to iterate a subset of a container, given two cursors, a start and end cursor.

For a discrete subtype, we can express in ada;

```
for I in 5 .. 10 loop
   ...
end loop
```

Where the bounds of iteration can be a subset of the values of a subtype.

It seems one ought to be able to do the same with cursors.

```
for Position in Cursor1 .. Cursor2 loop
   list (Position) := ...
end loop
```

Note: This currently isn't legal in Ada.

Otherwise, one has to write a while loop, which is a bit awkward.

```
Position : Cursor := Cursor1;
Iteration_Loop :
loop
   List (Position) := ...
   exit Iteration_Loop when
          Position := Cursor2;
   Next (Position);
end loop;
```

One can almost do this with the current containers, with Ada 2012 iterator syntax because they generally have an Iterate primitive that accepts a start cursor. e.g.

```
for I in List.Iterate(Start => Cursor1) loop
   List (I) := ...
end loop;
```

But we currently do not have Iterate primitives in the standard containers that also accept an End cursor, so these calls allow one to start somewhere in the middle of a container and iterate to the end, but not to stop earlier. It seems like it would be relatively easy to add such calls.

Then we could write;

```
for I in List.Iterate(Start => Cursor1,
         Finish => Cursor2) loop
   List (I) := ...
end loop;
```

Perhaps the intent was to use exit to exit the loop earlier when you hit the second cursor.

```
for I in List.Iterate(Start => Cursor1) loop
   List (I) := ...
   exit when I = Cursor2;
end loop;
```

I think that's somewhat satisfactory.

But I still think it would be nice to take this one step further and be able to add syntactic sugar to express this with range syntax, as shown in the example above.

I don't know how useful this would be, or how easy it would be to add to the language, but one place I would use it is when trying to iterate through containers in parallel where workers are each given a subset of the container by providing each worker with a start and end cursor.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 22 Dec 2017 17:45:22 -0600
Subject: Re: syntaxic exploration
Newsgroups: comp.lang.ada

[...]

> for I in List.Iterate(Start => Cursor1) loop

> List (I) := ...

> end loop;

This is true, too. But you just need to exit the loop at the end cursor to get the semantics you want:

```
for I in List.Iterate(Start => Cursor1) loop
   List (I) := ...
   exit when I = Cursor2;
end loop;
```

Ada doesn't care what kind of loop you use an exit in!

If there was to be a complaint here, it's that you can't easily use the element form ("of") with an ending cursor (because the cursor has to be explicit). But it seems weird to me to want to use cursors and yet hide them at the same time.

Conclusion: There isn't a need for an explicit "ending cursor", because an exit works fine for that in the cursor form. There still might be some sort of consistency argument, but we didn't add it originally since the exit is so easy to use for ending a loop "early".

Sudoku Solver Data Structures

From: Mace Ayres <mace.ayres@gmail.com>
Date: Fri, 29 Dec 2017 16:51:48 -0800
Subject: Arrays, slices, case, and 'in' strategies
Newsgroups: comp.lang.ada

I have a two dimensional array, Grid, 1 .. 9, 1 .. 9 index of Integer and array of a record type named cell and array is named grid. So, a grid is a 9x9 array of record type Cell. Cell has some properties, and functions in the containing package to enter, change, values in a field called Value. So I can traverse the array named grid with

```
for Row in 1 .. 9 loop
  for Column in 1 .. 9 loop
    Grid (Row, Column).x := y;
    -- x is some attribute of the cell/record/
    -- object that the array is type of
  end loop;   --and y is some value I assign
  -- to the x field, of the object cell in the
  -- array location (row, column)
end loop;
```

When attempting to assign a y value to the cell's x field, I want to check that the y that is to be assigned to the cell's x field does not already exist anywhere already in all the columns of the row, or all the rows of the target column. Easy enough with function:

```
Check_Row (R,C, Number : Integer ...
begin
  for C in 1 .. 9 loop  -- traverse the row in
  -- grid that is passed in with param r
    if Grid (R).Value = Number  --  if if
    -- grid(r,c).value = numb.
    -- numb is passed in also a param
   then  -- proposed value numb already
    -- exists in proposed row to put it in
      return false              -- not ok
...
```

function check_column uses some logic. Both work and are fast enough on my 64 bit Mac OS X.

Now, challenge is: besides checking to see if a proposed value (integer 1..0) already exists in the proposed row and column to put it in, I also have an abstraction of triads superimposed (mentally) on the grid, giving 9 triads layer on top of the 9x9 array called 'grid.' The 9x9 grid nicely holds 9 of these imaginary triads,, i.e., first 3 rows and first 3 columns are in triad 1 and first 3 rows and columns 4 to 6 are conceived as triad 2,

I need to check, in addition to whether the proposed integer value is already in any cells in that row, or column, whether the proposed value already exists in any cell.value field for any cell in triad. The in parameters of r,c can reveal what triad is in question, but it looks like some fat code to traverse the abstract 3x3 triads.

I know I could create some triad types of arrays, but I am wondering if I can slice the array grid into 9 collections and then check loop.. if numb (proposed number) already exists in the triad [this would be determined by row, column in parameters) OR also, the cell/record has a field with its triad, a constant property.

If this is too convoluted and or not clear, I can just delete the question, but if it's understandable, best approach recommended appreciated.

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Sat, 30 Dec 2017 09:34:41 +0100
Subject: Re: Arrays, slices, case, and 'in' strategies
Newsgroups: comp.lang.ada

Mace Ayres wrote:

[A problem description which sound very much like a part of a Sudoku solver. :-)]

You could make the two dimensional array a container with three different iterators, one for iterating over a selected column, one for iterating over a selected row, and one for iterating over a block.

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Sat, 30 Dec 2017 10:58:46 +0100
Subject: Re: Arrays, slices, case, and 'in' strategies
Newsgroups: comp.lang.ada

> [...]

You could create a function:

```
function Block (Row : Row_Number;
         Column : Column_Number) return
         Block_Number;
```

or a map:

```
type Block_Map is array (Row_Number,
         Column_Number) of
         Block_Number;
Block : constant Block_Map := ...;
```

both of which would be used as

```
Block (Row, Column)
```

From: Robert I. Eachus <rieachus@comcast.net>
Date: Sun, 31 Dec 2017 12:14:46 -0800
Subject: Re: Arrays, slices, case, and 'in' strategies
Newsgroups: comp.lang.ada

> [...]

I would do this by putting the Triad number in each cell, and having a set type:

```
type Sudoku_Set is array
         (Integer range 1 .. 9) of Boolean;
Reset : constant Sudoku_Set :=
         (others => False);
Set_Array : array (Integer range 1 .. 9)
         of Sudoku_Set;
Row_Array, Col_Array, Triad_Array :
         Set_Array := (others => Reset);
```

Now your tests for whether a number is already used is some ors:

```
if not (Row_Array (Row) or Col_Array
(Column) or Triad_Array (Cell (Row,
Column).Triad) (Candidate) then...
```

Oring the three arrays together then choosing a candidate should be faster than indexing them separately. I didn't attempt to compile the above so it probably includes a syntax error or two. The reason I didn't is that once you have the Ah Ha! moment of oring the sets together, you are likely to change your tree search to do the oring into a local variable, then do something like:

```
function Recur (Row, Column : in Integer)
is
  Local_Set : Sudoku_Set := Row_Array
         (Row) or Col_Array (Column) or
         Triad_Array (Cell (Row,
         Column).Triad;
begin
  if Cell(Row, Column) = 0 then
  -- set as part of conditions.
    if Row < 9 then
      return Recur (Row+1, Column);
    elsif Column < 9 then
      return Recur (1, Column+1);
    else
      return True;
    end if;
  end if;

  for I in Integer range 1 .. 9 loop
    if not Local_Set (I) then
      Grid (Row_Column).Value := I;
      Row_Array (Row) (I) := True;
      Col_Array (Column) (I) := True;
    end if;
    if Row < 9 then
      return Recur(Row+1, Column);
    elsif Column < 9 then
      return Recur (1, Column+1);
    else
      return True;
   end if;
  end loop;
end Recur;
```

This does a brute force search, but it should be fast enough. You can add bells and whistles like filling in a row column or triad cell when eight numbers have been used. I don't think that would make things any faster. Using an access value instead of an index for triads might make it faster, but that's the type of optimization best left until last.

From: Simon Wright <simon@pushface.org>
Date: Sat, 30 Dec 2017 17:41:53 +0000
Subject: Re: Arrays, slices, case, and 'in' strategies
Newsgroups: comp.lang.ada

> [...]

Oh, I see, it's Sudoku, and those are the given cells.

Anyway, this looks simple enough for checking whether the value is already in the block:

```
function Check_Block (
         For_Value : in Integer;
          At_Row : in Grid_Coordinate;
          At_Column : in Grid_Coordinate;
          In_Grid   : in Grid) return Boolean
is
  Row_First : constant Grid_Coordinate :=
         ((At_Row    - 1) / 3) * 3 + 1;
  Col_First : constant Grid_Coordinate :=
         ((At_Column - 1) / 3) * 3 + 1;
begin
  for Row in Row_First .. Row_First + 2
  loop
    for Col in Col_First .. Col_First + 2
    loop
      if In_Grid (Row, Col).X = For_Value
      then
        return False;
      end if;
    end loop;
  end loop;
  return True;
end Check_Block;
```

Common Part for all Exception Handlers

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Sat, 6 Jan 2018 23:04:09 +0100
Subject: Re: stopping a loop iteration without exiting it
Newsgroups: comp.lang.ada

Matt Borchers wrote:

> begin
>   ...
> exception
>   when EX1   => ...
>   when EX2   => ...
>   when others => ...
> finally
>   ...
> end;
>
> This would allow programmers to handle common cleanup operations that would otherwise have to be duplicated in every exception case.

Actually, it is possible (ever since Ada 83!) to have a common part for all exception handlers, and then different handlings according to the exception:

```
exception
  when others =>
    begin
      Common_Part;
      raise;
    exception
      when EX1 => ...
      when EX2 => ...
      ...
    end;
end;
```

On Submitting Proposals to the ARG

From: Simon Clubley <clubley@eisner.decus.org>
Date: Mon, 8 Jan 2018 21:48:51 -0000
Subject: Submitting requests to the ARG, was: Re: stopping a loop iteration without exiting it
Newsgroups: comp.lang.ada

> [...]

I don't know if Dmitry is joking or not, but language lawyers most certainly are _not_ required when submitting an issue to the ARG.

You just need to provide clear examples of your reasoning and maybe some suggestions about what you would like the new syntax or semantics to look like and why.

Most important however, is that you _MUST_ state at the beginning of your submission what it is that you think Ada is missing and what your proposal brings to Ada.

Everything else in your proposal should follow on from that; there's no point in submitting unstructured paragraphs of text to the ARG that don't state _what_ you are asking to be fixed in Ada and why.

Just make your submission readable, give _clear_ reasons for it, and provide examples that clarify your reasoning or concerns.

That's all I had to do. (Oh, and make sure you register for the ARG mailing list so that you can read the responses; the initial period after you submit your proposal is a two-way process where you can respond to questions and comments about your proposal.)

On Language Design

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 24 Jan 2018 14:39:30 +0100
Subject: Re: Five Years After
Newsgroups: comp.lang.ada

> [...]

Nothing happens without scientific efforts towards new concepts of language design. There was no any advance for more than 30 years. The compiler market was demolished long ago, so there is no commercial interest in any research.

Languages are designed either by hobbyists or by monopolists for purposes of customer lock and stiffening competition. They recycle old ideas good and bad all the same, just like fashion designers do their collections.

There is no push from the programmers' side either because there is no interest in software quality in general, quality does not sell.

Ergo, use Ada while you can.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 24 Jan 2018 20:44:21 -0600
Subject: Re: Five Years After
Newsgroups: comp.lang.ada

> [...]

This is one the best descriptions of modern language design that I've heard! Part of the problem is that few of these so called language designers have much exposure to many languages, so one doesn't really even know what does and does not work. And some features (exception contracts come to mind) are declared bad as much because of bad designs which inevitably lead to bad outcomes.

Ada is not immune to this, unfortunately. There is always pressure to add some pet feature from some other favorite language, and I have to hope that ultimately we will resist adding too much of that sort of stuff.

Random Numbers

From: Robert I. Eachus <rieachus@comcast.net>
Date: Fri, 2 Feb 2018 11:07:31 -0800
Subject: Re: Card game deck but with trumps suit for tarot "divination" Is there a better way than enumerating all cards?
Newsgroups: comp.lang.ada

> [...]

It also depends on whether you want selection with or without replacement. To deal out a deck you want dealing without replacement (no two hands get the Queen of Hearts, or whatever).

The best way to deal with replacement is to use a Long_Float random number generator then multiply by 78. Convert to Integer, and if 0, change to 78. (Doesn't matter, if you do that, what type of rounding is used.)

To shuffle a deck, make an array of 78 records with an Integer and a Long_Float. Assign the Integer field 1 through 78. Assign (78) random Long_Floats to the other field, then apply whatever sort algorithm you wish. (I tend to use a generic heap sort, but with only 78 values, you will spend more time setting things up and assigning random values than anything else.) You can either use these values to indirectly index into your cards, or just add the Long_Float field and let the sort shuffle the card values.

Why go through sorting the Long_Float values? If you draw without replacement, you draw from 78 cards, then 77, 76, 75,...4,3,2,1. There is nothing in Ada's random number generators to guarantee that these 78 generators work together to produce random values. By choosing 78 values from the same generator you don't have that problem.

Oh, what about assigning two equal (Long_Float) values to different cards? If you really worry about it, with more than 2**56 values, think of the card number as a second index, and if two values are equal you look at the card number. You don't do this, you just choose a sort that maintains the original order when two values are the same.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 2 Feb 2018 17:05:21 -0600
Subject: Re: Card game deck but with trumps suit for tarot "divination" Is there a better way than enumerating all cards?
Newsgroups: comp.lang.ada

> If you draw without replacement, you draw from 78 cards, then 77, 76, 75,...4,3,2,1. There is nothing in Ada's random number generators to guarantee that these 78 generators work together to produce random values. By choosing 78 values from the same generator you don't have that problem.

As previously noted, Ada 2020 has added an additional Random function into Discrete_Random to deal with this problem.

An alternative approach using current Ada would be to use a fixed Discrete_Random range (0 .. 77), and simply discard any values that identify cards already dealt, just retrying to get a random value. This works well and is properly uniform, but only if you are going to deal part of the deck (say 25%). If you need to deal the whole deck, it can get slow toward the end when most of the values returned from Random have already been dealt. (That's especially bad if you forget to special case the last card to the dealt - you don't need a random choice to figure out which one that is after all of the rest have been dealt.)

Experience shows that most users don't get this right, as did the discussion on the topic (in which many knowledgeable people suggested approaches which just don't work). I had to fix the Janus/Ada random number generator after that discussion, as it had made one of the mistakes as well.

From: Robert I. Eachus <rieachus@comcast.net>
Date: Sat, 3 Feb 2018 08:59:24 -0800
Subject: Re: Card game deck but with trumps suit for tarot "divination" Is there a better way than enumerating all cards?
Newsgroups: comp.lang.ada

> [...]

If your random number generator uses the Linux RNG for initialization, and has a long enough period (the GNAT generator does) the above will work, and is reasonably fast. Throwing back duplicates, as Randy said, can take a lot of RNG calls near the end. If you are going to special case the last card, why not special case the last few cards, generate a permutation of say six values and use that for the last six cards. Much faster, but the one value per card, then sort takes O(n log2 n) time, and is usually dominated by the n RNG calls.

Thoughts on Access Types

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Fri, 9 Feb 2018 18:01:38 +0100
Subject: Re: grassroots thoughts on access types
Newsgroups: comp.lang.ada

Others have responded to your questions. I'd like to mention my thoughts on using access-to-object types (which I'll call "access types") in general. (By "use", I'm referring to designing code with access types, not to using them when required to in order to reuse existing code. Gnoga, for example, requires you to supply some access values to use it.)

It's very rare to actually need to use access types in Ada. Other than while learning the language, it's quite likely that you'll never encounter a situation in which you'll need them. So while it's important to learn how they work, you shouldn't be designing in terms of them.

My personal rules for designing with access types:

1. Don't use access types
2. If you think you need access types, see rule 1.
3. If you still think you need access types, don't use visible or anonymous access types
4. If you think you need visible or anonymous access types, see rule 3.
5. If you still think you need visible or anonymous access types, then you shouldn't design software.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 9 Feb 2018 18:19:27 +0100
Subject: Re: grassroots thoughts on access types
Newsgroups: comp.lang.ada
> [...]

I mostly agree with that, yet there are some notable exceptions from these rules.

a. Dispatching operation. It cannot have a named access type. The conflicting design rule here is that *all* operations must be dispatching and any type must have a class.

b. Anonymous access type accepts arguments without explicit type conversion. Type conversions are always bad, ones exposing run-time hazard are more than bad.

c. Mix-in discriminant, anonymous access type is the only way in many cases. Mix-in itself is a horrid design pattern, but there is no multiple inheritance here to replace mix-in and no proper classes of protected and task types either.

In general, anonymous access type is the only way to enforce referential semantics. It would be great if Ada had it decoupled that from pointers. Unfortunately it did not.

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Fri, 9 Feb 2018 20:12:02 +0100
Subject: Re: grassroots thoughts on access types
Newsgroups: comp.lang.ada

> [...]

Anything that requires the use of anonymous access types is bad and should not be used.

From: Robert A Duff <bobduff@TheWorld.com>
Date: Fri, 09 Feb 2018 15:17:06 -0500
Subject: Re: grassroots thoughts on access types
Newsgroups: comp.lang.ada

> Anything that requires the use of anonymous access types is bad and should not be used.

So you won't use things like "A(X) := A(X) + 1;", where A is a Vector?

Or "for X of A loop..."?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 9 Feb 2018 23:06:06 +0100
Subject: Re: grassroots thoughts on access types
Newsgroups: comp.lang.ada

[...]

Having said that, nether iteration interface nor indexing interface should rely on access types, anonymous or not. It is a language design flaw that they do. Nevertheless both interfaces are clearly useful and desirable.

Anonymous access type is an ugly hack to work around real language problems. It should have never been introduced. The problems should have been properly addressed instead.

Finding Memory Leaks

From: Tomasz "darkestkhan" Maluszycki <darkestkhan@gmail.com>
Date: Sun, 11 Feb 2018 02:58:33 -0800
Subject: Re: How to optimize use of RAM/disk access ?
Newsgroups: comp.lang.ada

> [...]

Use valgrind for checking memory leaks - top would show you only big ones (I know that few bytes may not sound like much, but IT IS A LOT - for example windows 10 audio driver seems to leak memory [otherwise you can't explain why it uses over 500MB of memory after just 2-3 weeks of running] - how? small leak over prolonged period of time becomes big one).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 11 Feb 2018 12:38:11 +0100
Subject: Re: How to optimize use of RAM/disk access ?
Newsgroups: comp.lang.ada

> [...]

For memory leaks I would recommend gnatmem. It is extremely easy to use. No code change required. It groups similar memory allocations together. It also finds leaks related to C code, which is the main source of leaks when using bindings to low-level C libraries.

Default_Storage_Pool

From: Simon Belmont <sbelmont700@gmail.com>
Date: Tue, 20 Feb 2018 18:00:05 -0800
Subject: Default_Storage_Pool
Newsgroups: comp.lang.ada

Can anyone offer insight into what exactly should happen when Default_Storage_Pool is explicitly set within an extended return statement to a pool within the return object? In particular, consider these shenanigans:

```
package O is
  pool_1 : My_Fancy_Pool
  pragma Default_Storage_Pool (pool_1);

  type T is limited
    record
      pool_2 : My_Fancy_Pool
      p1     : access Integer;
    end record;

  function F return T;
end O;

package body O is
  function F return T is
  begin
    return Result : T do
      declare
        pragma Default_Storage_Pool
         (Result.pool_2);  -- legal?
        p2 : access Integer;
      begin
        p2 := new integer'(42);
        Result.p1 := new integer'(43);
      end;
    end return;
  end F;
end O;
```

GNAT happily accepts this, but based on print lines, it allocates Result.p1 from pool_1 and p2 from some unspecified default pool (i.e. neither pool_1 or pool_2). I wasn't sure what I was expecting; I assumed an error message, but failing that, both to go into result.pool_2, and was surprised to get neither.

Any clarifications are appreciated.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 21 Feb 2018 19:13:58 -0600
Subject: Re: Default_Storage_Pool
Newsgroups: comp.lang.ada

> [...]

Default_Storage_Pool only has an effect when an access type is declared (not when it is used). Thus the allocators for the component of type T should use Pool_1 regardless of what the Default_Storage_Pool is when the allocator is written.

But I'd expect the local anonymous access allocator to use Pool_2. I don't see any reason to use some other pool in this case - 13.11.3(6-6.3/4) is pretty clear about this, and the rules specifically were designed so that it would apply to anonymous access types.

Thus, this appears to be a bug, but I also fail to see any use for it (the anonymous access type having to disappear long before anyone can used the return value), so I would probably not give it much priority if it was reported to me. (Of course, a fuller example could cause me to change my mind on that.)

From: Simon Belmont <sbelmont700@gmail.com>
Date: Thu, 22 Feb 2018 05:02:42 -0800
Subject: Re: Default_Storage_Pool
Newsgroups: comp.lang.ada

> [...]

Thank you, that was what I thought it should do. I had no legitimate use case, I was just trying to tease out counter-examples to confirm my understanding.

I claim no in-depth comprehension, but because of either subsequent bugs or perhaps 2012 changes to AATs (or perhaps legitimately), GNAT also takes this:

```
package body O is
  janky : access Integer;

  function F return T is
  begin
    return Result : T do
      declare
        pragma Default_Storage_Pool
         (Result.pool_2);
        p2 : access Integer;
      begin
        p2 := new integer'(42);
        janky := p2;  --legal?
      end;
    end return;
  end F;
end O;
```

When does janky become a dangling pointer? Surely if the default pool is local to F (because the entire pool goes away after F ends), never if the default pool is pool_1 (since it has the same lifetime as janky), and 'possibly' if it's contained within Result.pool_2 (since it depends on where the client has creates the result).

GNAT accepts all three variations (well, subject to the aforementioned bug presumably making case 3 the same as 1) and raises no exceptions for any of them.

Thank you for the continued explanations

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 22 Feb 2018 18:06:10 -0600
Subject: Re: Default_Storage_Pool
Newsgroups: comp.lang.ada

> [...]

I think the assignment to Janky should raise Program_Error, as it can take an object of any lifetime that is *longer* than itself.

RM 3.10.2(13.3/4) says this explicitly "...; accessibility checks ensure that this is never deeper than that of the declaration of the stand-alone object".

Trying to figure out precisely what check this is talking about is hard, however (the AARM is supposed to explain that part as it is claimed to follow from other rules -- but it doesn't). I'm not going to try to work that out in detail (it's also possible that it is supposed to be illegal).

Diamond Diagram for 'with'

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 21 Feb 2018 17:20:33 +0100
Subject: Diamond diagram for 'with'
Newsgroups: comp.lang.ada

I would ask language lawyers regarding multiple with.

Consider this:

```
limited with Root.A;
package Root is
end Root;

package Root.A is
  type T is ...;
end Root.A;

with Root.A;
package Root.B is
end Root.B;
```

Now Root.B has both limited (inherited from Root) and full "with" of Root.A. So, may Root.B use Root.A.T? It cannot according to "limited with" and it can due to full "with". Which one to win?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 21 Feb 2018 19:38:24 -0600
Subject: Re: Diamond diagram for 'with'
Newsgroups: comp.lang.ada

> [...]

Off the top of my head, it should be the "full with". Generally, Ada allows one to open more visibility, but you can't remove it. Deriving this formally would be somewhat painful, but it has to be true, since the motivating use for limited with is something like:

```
limited with P;
package Q is
  ...
end Q;

limited with Q;
package P is
  ...
end P;

with Q;
package body P is
  -- Q is normally visible here.
end P;
```

If the limited with "won" in the body, one could never access the full package in the body, which would make actually implementing any mutually dependent package hard. (You may want to call some primitive routine declared in Q in the body of P, but that isn't possible for a "limited with").

Since all withs work basically the same way, the same has to be true for a child package.

BTW, Root.A.T would be legal either way. But if it came from the limited with, it would be an incomplete type (which has its own limitations).

Use of Separates

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 28 Feb 2018 17:23:30 -0600
Subject: Re: body stub not allowed in inner scope
Newsgroups: comp.lang.ada

> [...]

Does anyone other than ACATS tests actually use stubs these days? Why? [...]

From: Simon Wright <simon@pushface.org>
Date: Thu, 01 Mar 2018 08:24:55 +0000
Subject: Re: body stub not allowed in inner scope
Newsgroups: comp.lang.ada

> [...]

My use case for stubs is that I have a code generator that transforms a UML model into the framework of an Ada solution; subprograms, task and protected type bodies are generated as separates, so that there's no hassle with having the tool work out how not to overwrite the real bodies. This was triggered by bad experiences with Rational Rose back in 2000.

I have to admit that the only commercial project that I'm aware of that uses this tool is (if any work is being done at all) in very low-level long term maintenance, and (on past evidence) supremely unlikely to be interested in upgrading to a new Ada standard.

[I have one separate package body of 300 lines - to reduce clutter in a package body that was already 260 lines. Even if emacs ada-mode would now be able to fold it, it certainly couldn't then!]

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Thu, 01 Mar 2018 21:52:03 +0100
Subject: Re: body stub not allowed in inner scope
Newsgroups: comp.lang.ada

> My use case for stubs is that I have a code generator that transforms a UML model into the framework of an Ada solution [...]

This sounds rather similar to some work I've done for Consafe Logistics last year. I wrote a code generator, which compiles Swagger/OAS specifications of REST interfaces into packages declaring the services and the types used. The actual service implementations are made separate, to have the optimal separation of developer-written and tool-written Ada code.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 1 Mar 2018 16:45:26 -0600
Subject: Re: body stub not allowed in inner scope
Newsgroups: comp.lang.ada

> [...] The actual service implementations are made separate, to have the optimal separation of developer-written and tool-written Ada code.

The CLAW Builder generated code does this, but used user-generated packages for this purpose (rather than separates). The user provides the name of the package, and the generated code makes the needed calls. We were about to allow the user to choose to generate the package specifications for such packages (that never got implemented) so that the bodies would be easier to write (that would make the profiles of the needed routines obvious, rather than just being in the popup help for the builder). We never even considered subunits for this purpose (maybe should have? Dunno.) One never wants to mix machine-generated code (which the user should look at only in cases where there is a bug in the tools) with user-generated code (which the user obviously has to manage, and would prefer to use their normal IDE for development).

Alternatives to Unrestricted_Access

From: Jere <jhb.chat@gmail.com>
Date: Wed, 7 Mar 2018 12:11:01 -0800
Subject: Ada Alternatives to Unrestricted_Access
Newsgroups: comp.lang.ada

I'm currently updating a generic package that takes a container type as an input and creates an iterative type around it. I found quite a few instances of GNAT's Unrestricted_Access which appear to be necessary because of the Iterate function's container parameter of being mode "in" and the iterator needing a non constant access to it to supply tamper check semantics. I tried changing the mode to "in out" but that caused a slew of errors where Iterate was used inside other operations that had the Container supplied via "in" mode parameters. Changing those other operations doesn't make sense (and causes other errors later), but I don't like relying on GNAT's implementation defined attribute if I can help it.

I was able to use System.Address_To_Access_Conversions to achieve both compiler happiness and expected runtime execution, but I am worried this is not legal as it is skirting kind of close to Address overlays. In this case the types are identical aside from one (the Iterate "in" mode parameter) being constant and the other (the iterators "Container_Access" component) being variable. However, I am not fully convinced it is (relatively) safe or portable yet.

The gist of the differences being:

```
Container_Access =>
        C'Unrestricted_Access
```

is converted to:

```
Container_Access =>
        A2A.To_Pointer(C'Address)
```

where:

```
   package A2A is new
System.Address_To_Access_Conversions
(Container);
```

and the parameter mode of C is defined as:

```
function Iterate(C : in Container) return
        Iterator_Interfaces.
        Forward_Iterator'Class;
```

and the generic formal for Container

```
type Container is tagged private;
```

I'm working with an existing code base, so complete overhaul is not a great option.

If it isn't a good idea, are there alternatives? Since the Container implementation is unknown in the generic, I wasn't able to get a version of the Rosen technique to work as an alternative (same constant vs variable access problem).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 7 Mar 2018 21:38:07 +0100
Subject: Re: Ada Alternatives to Unrestricted_Access
Newsgroups: comp.lang.ada

> [...]

I am not sure I understand the problem. Why don't you pass a function Self along with the container type:

```
generic
   type Container is tagged whatever;
   function Self (C : Container'Class)
      return not null access
         Container'Class;
package Iterative_Mess is
```

From: Jere <jhb.chat@gmail.com>
Date: Wed, 7 Mar 2018 14:29:46 -0800
Subject: Re: Ada Alternatives to Unrestricted_Access
Newsgroups: comp.lang.ada

> [...]

Because the package is already in use in other code, so changing the specification would be a very time consuming change. I was just trying to get rid of the Unrestricted_Access if I could and was asking if using Address_To_Access_ Conversions was an ok way to do so (is it portable Ada, and is it defined behavior) OR if there was a different alternative.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 8 Mar 2018 09:27:01 +0100
Subject: Re: Ada Alternatives to Unrestricted_Access
Newsgroups: comp.lang.ada

> [...]

You can default Self when not used.

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Wed, 7 Mar 2018 19:18:45 -0800
Subject: Re: Ada Alternatives to Unrestricted_Access
Newsgroups: comp.lang.ada

I had the same problem for GL[U[T]] bindings (work on both GNAT and ObjectAda, and perhaps other Ada systems) [1]. A quick search with AZip (shameless plug :-) [2] leads to

```
 package A2A_double is new System.
         Address_To_Access_Conversions
         (Double);
  procedure Vertex (v: Double_Vector_3D)
  is
    begin
   Vertex3dv(
         A2A_double.To_Pointer(
         v(0)'Address));
    -- This method is functionally identical
    -- to using GNAT's 'Unrestricted_Access
  end Vertex;
```

Other places use an Unchecked_Conversion.

The GNAT doc says:

"The Unrestricted_Access attribute is similar to Access except that all accessibility and aliased view checks are omitted. This is a user-beware attribute. It is similar to Address, for which it is a desirable replacement where the value desired is an access type. In other words, its effect is identical to first applying the Address attribute and then doing an unchecked conversion to a desired access type."

[1] https://globe3d.sourceforge.io/

[2] https://azip.sourceforge.io/

Revolutionize your software verification

Efficiency, Automation, Reliability

RAPITA Systems Ltd
A DANLAW Company

Unit testing · System testing · Coverage analysis · Timing analysis
V&V services · Multicore timing services · DO-178C training
Ada · C · C++

www.rapitasystems.com

Conference Calendar

Dirk Craeynest

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

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

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

2018

☺ April 09-12 — **The Art, Science, and Engineering of Programming Conference** (Programming'2018), Nice, France. Topics include: everything to do with programming, including the experience of programming; general-purpose programming; distributed systems programming; parallel and multi-core programming; security programming; interpreters, virtual machines and compilers; modularity and separation of concerns; model-based development; testing and debugging; program verification; programming education; programming environments; etc.

- April 09 — 2nd **Workshop on Modern Language Runtimes, and Ecosystems** (MoreVMS'2018). Topics include: design, implementation, and usage of modern languages and runtimes; interoperability between languages; tooling support (e.g. debugging, profiling, etc.); programming language development environments; case studies of existing language implementation approaches; language implementation challenges and trade-offs; surveys and usage reports to understand usage in the wild; ideas for how we should build languages in the future; etc.

April 09-13 — 33rd ACM **Symposium on Applied Computing** (SAC'2018), Pau, France.

- ☺ April 09-13 — **Track on Object-Oriented Programming Languages and Systems** (OOPS'2018). Topics include: aspects and components; code generation, and optimization; distribution and concurrency; evaluation; formal verification; Internet of Things technology and programming; integration with other paradigms; interoperability, versioning and software evolution and adaptation; language design and implementation; modular and generic programming; runtime verification and monitoring; safe, secure and dependable software; static analysis; testing and debugging; type systems; etc.
- ☺ April 09-13 — **Track on Programming Languages** (PL'2018). Topics include: compiling techniques, domain-specific languages, garbage collection, language design and implementation, languages for modeling, model-driven development, new programming language ideas and concepts, practical experiences with programming languages, program analysis and verification, programming languages from all paradigms, etc. Deadline for submissions: September 25, 2016 (full papers).
- April 09-13 — 13th **Track on Dependable, Adaptive, and Trustworthy Distributed Systems** (DADS'2018). Topics include: Dependable, Adaptive, and trustworthy Distributed Systems (DADS); middleware for DADS; modeling, design, and engineering of DADS; foundations and formal methods for DADS; etc.
- April 12 — **Track on Software Verification and Testing** (SVT'2018), 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, 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.

April 09-13 9th ACM/SPEC **International Conference on Performance Engineering** (ICPE'2018), Berlin, Germany. Theme: "Continuous Performance Assurance in Agile Delivery".

April 09-13 11th IEEE **International Conference on Software Testing, Verification and Validation** (ICST'2018), Västerås, Sweden. Topics include: experience reports, formal verification, model checking, security testing, software reliability, testing in specific domains (such as embedded, concurrent, distributed, real-time, ..., systems), testing/debugging tools, etc.

April 10-13 11th **Cyber-Physical Systems Week** (CPS Week'2018), Porto, Portugal.

☺ April 11-13 24th IEEE **Real-Time and Embedded Technology and Applications Symposium (**RTAS'2018), Porto, Portugal. In conjunction with CPSWeek'2018. Topics include: timing issues ranging from traditional hard real-time systems to latency-sensitive systems with soft real-time requirements; original systems and applications, case studies, methodologies and applied algorithms that contribute to the state of practice in the design, implementation and verification of real-time systems; embedded, networked and cyber-physical systems that consider real-time aspects; etc.

* {PT} 2018/04/11-13: 9th ACM/IEEE **International Conference on Cyber-Physical Systems** (ICCPS'2018), Porto, Portugal. In conjunction with CPSWeek'2018. Topics include: development of technologies, tools, and architectures for building CPS systems; design, implementation, and investigation of CPS applications; secure and resilient CPS infrastructure; etc.

April 14-20 21st **European Joint Conferences on Theory and Practice of Software** (ETAPS'2018), Thessaloniki, Greece. Events include: ESOP (European Symposium on Programming), FASE (Fundamental Approaches to Software Engineering), FoSSaCS (Foundations of Software Science and Computation Structures), POST (Principles of Security and Trust), TACAS (Tools and Algorithms for the Construction and Analysis of Systems), SV-COMP (7th Competition on Software Verification).

April 17-19 10th **NASA Formal Methods Symposium** (NFM'2018), Newport News, Virginia, USA. Topics include: identify challenges and provide solutions for achieving assurance for critical systems; model checking, static analysis, use of formal methods in software and system testing, compositional techniques, parallel and/or distributed techniques, safety cases and system safety, fault tolerance, model-based development, etc. Deadline for registration: April 1, 2018.

♦April 18-20 19th **International Real-Time Ada Workshop** (IRTAW'2018), Benicàssim, Spain. In cooperation with Ada-Europe.

April 23-27 21st **Ibero-American Conference on Software Engineering** (CIbSE'2018), Bogotá, Colombia. Event includes Software Engineering Track (SET) and Experimental Software Engineering Track (ESELAW).

April 30 - May 04 2nd **International Conference on Software Architecture** (ICSA'2018), Seattle, USA. Topics include: model driven engineering for continuous architecting; component based software engineering and architecture design; re-factoring and evolving architecture design decisions and solutions; architecture frameworks and architecture description languages; preserving architecture quality throughout the system lifetime; software architecture for legacy systems and systems integration; architecting families of products; software architects roles and responsibilities; training, education, and certification of software architects; industrial experiments and case studies; bold arguments against current research directions and results; results that challenge established results or beliefs giving evidence that call for fundamentally new directions, open up new research avenues where software architecture research can contribute; etc.

May 21-23 17th **International Conference on Software Reuse** (ICSR'2018), Madrid, Spain. Theme: "New Opportunities for Software Reuse". Topics include: component-based reuse techniques, generative reuse, systematic reuse approaches helping industries transitioning from ad-hoc approaches, reverse engineering of potentially reusable components, evolution and maintenance of reusable assets, development of reusable components for Product Line Engineering, software variability approaches for configuring and deriving reusable assets, dynamic aspects of reuse (i.e. post-deployment time), etc.

May 21-25 19th **International Conference on Agile Software Development** (XP'2018), Porto, Portugal.

May 21-25 32nd IEEE **International Parallel and Distributed Processing Symposium** (IPDPS'2018), Vancouver, Canada.

May 27-29 — 13th IEEE/ACM **International Conference on Global Software Engineering** (ICGSE'2018), Gothenburg, Sweden.

☺ May 29-31 — 21st IEEE **International Symposium On Real-Time Computing** (ISORC'2018), Singapore. Topics include: object/component/service-oriented real-time distributed computing (ORC) technology, programming and system engineering (real-time programming challenges, ORC paradigms, languages, ...), trusted and dependable systems, system software (real-time kernels, middleware support for ORC, extensibility, synchronization, scheduling, fault tolerance, security, ...), applications (medical devices, intelligent transportation systems, industrial automation systems, Internet of Things and Smart Grids, embedded systems in automotive, avionics, consumer electronics, ...), system evaluation (performance analysis, monitoring & timing, dependability, fault detection and recovery time, ...), cyber-physical systems, etc.

May 27 - June 06 — 40th **International Conference on Software Engineering** (ICSE'2018), Gothenburg, Sweden.

June 11-15 — 30th **International Conference on Advanced Information Systems Engineering** (CAiSE'2018), Tallin, Estonia. Theme: "Information Systems in the Big Data Era". Topics include: methods, models, techniques, architectures and platforms for supporting the engineering and evolution of information systems and organizations in the big data era.

♦ June 18-22 — 23rd **International Conference on Reliable Software Technologies - Ada-Europe'2018**. Lisbon, Portugal. Sponsored by Ada-Europe, in cooperation with ACM SIGAda, SIGBED, SIGPLAN, and the Ada Resource Association (ARA).

June 25-29 — **Software Technologies: Applications and Foundations** (STAF'2018), Toulouse, France. Successor of the TOOLS federated event. Topics include: practical and foundational advances in software technology, such as object-oriented design, testing, formal approaches to modelling and verification, transformation, model-driven engineering, aspect-oriented techniques, and tools. Deadline for submissions: April 20, 2018 (workshop papers).

- June 27-29 — 16th **International Conference on Software Engineering and Formal Methods** (SEFM'2018). Topics include: light-weight and scalable formal methods; software evolution, maintenance, re-engineering and reuse; programming languages; abstraction and refinement; correctness-by-construction; model checking; verification and validation; testing; safety-critical, fault-tolerant and secure systems; software certification; real-time and embedded systems; application and technology transfer; case studies, best practices and experience reports; tool integration; education; etc.
- June 27-29 — 12th **International Conference on Tests And Proofs** (TAP'2018). Topics include: many aspects of verification technology, including foundational work, tool development, and empirical research; the connection between proofs (and other static techniques) and testing (and other dynamic techniques); verification and analysis techniques combining proofs and tests; program proving with the aid of testing techniques; deductive techniques supporting the automated generation of test vectors and oracles; deductive techniques supporting novel definitions of coverage criteria; program analysis techniques combining static and dynamic analysis; testing and runtime analysis of formal specifications; verification of verification tools and environments; applications of test and proof techniques in new domains, such as security, configuration management, learning; combined approaches of test and proof in the context of formal certifications (Common Criteria, CENELEC, ...); case studies, tool and framework descriptions, and experience reports about combining tests and proofs; etc.

June 25-29 — 12th ACM **International Conference on Distributed Event-Based Systems** (DEBS'2018), Hamilton, New Zealand. Topics include: systems dealing with detecting, processing and responding to events and with massively distributed middleware and applications, real-time analytics, complex event processing, distributed programming, fault tolerance, reliability, availability, scalability, internet of things, cyber-physical systems, transportation, enterprise application integration, etc.

June 27-29 — 22nd **International Conference on Evaluation and Assessment in Software Engineering** (EASE'2018), Christchurch, New Zealand.

July 03-06 — 30th **Euromicro Conference on Real-Time Systems** (ECRTS'2018), Barcelona, Spain. Topics include: all aspects of real-time systems, such as scheduling design and analysis, real-time operating systems,

	hypervisors and middlewares, virtualization and timing isolation, mixed-criticality design & assurance, worst-case execution time analysis, modelling and/or formal methods, industrial use-cases and real-time applications, tools, compilers and benchmarks for embedded systems, etc. Event includes: Worst-Case Execution Time analysis (WCET), Workshop on Analysis Tools and methodologies for Embedded and Real-time Systems (WATERS).
☺ July 09-10	**Workshop: Konstruktion von SafeWare - Construction of SafeWare** (KSW'2018), Karlsruhe, Germany. Co-organized by Ada-Deutschland.
July 14-17	30th **International Conference on Computer-Aided Verification** (CAV'2018), Oxford, UK. Topics include: theory and practice of computer-aided formal analysis methods for hardware and software systems, algorithms and tools for verifying models and implementations, specifications and correctness criteria for programs and systems, deductive verification using proof assistants, program analysis and software verification, formal methods for cyber-physical systems, verification methods for parallel and concurrent systems, testing and run-time analysis based on verification technology, applications and case studies in verification and synthesis, verification in industrial practice, formal models and methods for security, etc.
July 15-17	22nd **International Symposium on Formal Methods** (FM'2018), Oxford, UK. Topics include: formal methods for the engineering of computer-based systems and software; such as industrial applications of formal methods; experience with formal methods in industry; tool usage reports; advances in automated verification, model-checking, and testing with formal methods; tools integration; environments for formal methods; development processes with formal methods; usage guidelines for formal methods; etc.
July 16-20	18th IEEE **International Conference on Software Quality, Reliability and Security** (QRS'2018), Lisbon, Portugal. Topics include: reliability, security, availability, and safety of software systems; software testing, verification and validation; program debugging and comprehension; fault tolerance for software reliability improvement; modeling, prediction, simulation, and evaluation; metrics, measurements, and analysis; software vulnerabilities; formal methods; benchmark, tools, and empirical studies; etc. Deadline for submissions: April 1, 2018 (workshop papers), May 1, 2018 (Student Doctoral Program, fast abstracts, industry track).
☺ July 16-22	32nd **European Conference on Object-Oriented Programming** (ECOOP'2018), Amsterdam, the Netherlands.
July 23-27	42nd **Annual** IEEE **Conference on Computer Software and Applications** (COMPSAC'2018), Tokyo, Japan. Deadline for submissions: April 10, 2018 (workshop papers).
July 25-28	37th ACM **Symposium on Principles of Distributed Computing** (PODC'2018), Royal Holloway, University of London, UK.
August 29-31	44th **Euromicro Conference on Software Engineering and Advanced Applications** (SEAA'2018), Prague, Czech Republic. Topics include: information technology for software-intensive systems; conference tracks on DSLs and Model-Based Development (DSLMBD), Software Process and Product Improvement (SPPI), etc.; tentative special sessions on Cyber-Physical Systems (CPS), Software Engineering and Technical Debt (SEaTeD), Monitoring Large-Scale Software Systems (MoLS), etc.
August 29-31	12th **International Symposium on Theoretical Aspects of Software Engineering** (TASE'2018), Guangzhou, China. Topics include: theoretical aspects of software engineering, such as abstract interpretation, component-based software engineering, cyber-physical systems, distributed and concurrent systems, embedded and real-time systems, formal verification and program semantics, integration of formal methods, language design, model checking and theorem proving, model-driven engineering, object-oriented systems, program analysis, reverse engineering and software maintenance, run-time verification and monitoring, software architectures and design, software testing and quality assurance, software safety, security and reliability, specification and verification, type systems, tools exploiting theoretical results, etc.
September 03-07	33rd IEEE/ACM **International Conference on Automated Software Engineering** (ASE'2018), Montpellier, France. Topics include: foundations, techniques, and tools for automating the analysis, design, implementation, testing, and maintenance of large software systems; maintenance and evolution; model-driven development; reverse engineering and re-engineering; specification languages; software analysis; software architecture and design; software product line engineering; software security and trust; testing, verification, and validation; etc. Deadline for submissions: April 19, 2018 (abstracts),

April 26, 2018 (papers), April 30, 2018 (tutorials), May 31, 2018 (tool demos, doctoral symposium), June 15, 2018 (journal-first submissions).

September 04-06 4th **Symposium on Dependable Software Engineering: Theories, Tools and Applications** (SETTA'2018), Beijing, China. Topics include: formalisms for modeling, design and implementation; model checking, theorem proving, and decision procedures; scalable approaches to formal system analysis; 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, and cyber-physical systems; mixed-critical applications and systems; safety, reliability, robustness, and fault-tolerance; applications and industrial experience reports; tool integration; etc.

September 05-07 14th **International Conference on integrated Formal Methods** (iFM'2018), Maynooth, Ireland. Topics include: hybrid approaches to formal modeling and analysis; i.e., the combination of (formal and semi-formal) methods for system development, regarding both modeling and analysis, and covering all aspects from language design through verification and analysis techniques to tools and their integration into software engineering practice. Deadline for submissions: April 16, 2018 (abstracts), April 20, 2018 (papers).

September 09-12 **Federated Conference on Computer Science and Information Systems** (FedCSIS'2018), Poznan, Poland. Event includes: 3rd International Workshop on Language Technologies and Applications (LTA), Joint 38th IEEE Software Engineering Workshop and 5th International Workshop on Cyber-Physical Systems (SEW & IWCPS), etc. Deadline for submissions: May 15, 2018 (papers), June 12, 2018 (position papers).

Sep 30 - Oct 5 **Embedded Systems Week** 2018 (ESWEEK'2018), Torino, Italy. Topics include: all aspects of embedded systems and software. Includes CASES'2018 (International Conference on Compilers, Architectures, and Synthesis for Embedded Systems), CODES+ISSS'2018 (International Conference on Hardware/Software Codesign and System Synthesis), EMSOFT'2018 (International Conference on Embedded Software). Deadline for submissions: April 3, 2018 (journal track full papers, tutorials), May 30, 2018 (work-in-progress track papers).

Sep 30 - Oct 5 **International Conference on Compilers, Architecture, and Synthesis for Embedded Systems** (CASES'2018). Topics include: the latest advances in compilers and architectures for high-performance, low-power embedded systems; leading edge research in embedded processor, memory, interconnect, storage architectures and related compiler techniques targeting performance, power, security, reliability, predictability issues for both traditional and emerging application domains; innovative papers addressing design, synthesis & optimization challenges in heterogeneous, accelerator-rich architectures. Deadline for submissions: April 3, 2018 (full papers), May 30, 2018 (Work-in-Progress papers).

Sep 30 - Oct 5 ACM SIGBED **International Conference on Embedded Software** (EMSOFT'2018). Topics include: the science, engineering, and technology of embedded software development; research in the design and analysis of software that interacts with physical processes; results on cyber-physical systems, which compose computation, networking, and physical dynamics. Deadline for submissions: April 3, 2018 (full papers).

☺ October 10-12 26th **International Conference on Real-Time Networks and Systems** (RTNS'2018), Poitiers, France. Topics include: real-time applications design and evaluation (automotive, avionics, space, railways, telecommunications, process control, multimedia), real-time aspects of emerging smart systems (cyber-physical systems and emerging applications, ...), real-time system design and analysis (real-time tasks modeling, task/message scheduling, mixed-criticality systems, Worst-Case Execution Time (WCET) analysis, ...), software technologies for real-time systems (model-driven engineering, programming languages, compilers, WCET-aware compilation and parallelization strategies, middleware, Real-time Operating Systems (RTOS), hypervisors), formal specification and verification, real-time distributed systems (fault tolerance, task/messages allocation, ...), etc. Deadline for paper submissions: June 26, 2018.

October 11-12 12th ACM/IEEE **International Symposium on Empirical Software Engineering and Measurement** (ESEM'2018), Oulu, Finland. Topics include: the strengths and weaknesses of software engineering technologies and methods from a strong empirical viewpoint, including quantitative, qualitative, and mixed studies; case studies, action research, and field studies; replication of empirical studies and families of studies; mining software engineering repositories; empirically-based decision making; assessing the benefits/costs associated with using certain development technologies; industrial experience, software project experience, and knowledge management; software technology transfer to industry; etc. Deadline for submissions: May 18, 2018 (full paper abstracts), May 25, 2018 (full papers), July 1, 2018 (Emerging Results, Vision papers), July 20, 2018 (industrial papers, posters), August 10, 2018 (Journal-First track).

☺ November 04-09 ACM **Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2018), Boston, Massachusetts, USA. Topics include: all aspects of software construction, at the intersection of programming, languages, and software engineering. Deadline for submissions: April 16, 2018 (PACMPL issue OOPSLA), April 23, 2018 (Onward! papers, Onward! essays), June 29, 2018 (GPCE abstracts - Generative Programming: Concepts & Experiences, SLE abstracts - Software Language Engineering), July 1, 2018 (DLS - Dynamic Languages Symposium), July 6, 2018 (GPCE - Generative Programming: Concepts & Experiences, SLE - Software Language Engineering), July 20, 2018 (Doctoral Symposium), July 27, 2018 (Student Research Competition), September 22, 2018 (Posters), end of September 2018 (Student Volunteers applications).

November 5-6 11th ACM **SIGPLAN International Conference on Software Language Engineering** (SLE'2018). Topics include: areas ranging from theoretical and conceptual contributions, to tools, techniques, and frameworks in the domain of software language engineering; generic aspects of software languages development rather than aspects of engineering a specific language; software language design and implementation; software language validation; software language integration and composition; software language maintenance (software language reuse, language evolution, language families and variability); domain-specific approaches for any aspects of SLE (design, implementation, validation, maintenance); empirical evaluation and experience reports of language engineering tools (user studies evaluating usability, performance benchmarks, industrial applications); etc. Deadline for submissions: June 29, 2018 (abstracts), July 6, 2018 (papers), August 31, 2018 (artifacts).

November 04-09 12th **Joint European Meeting of the Software Engineering Conferenc**e and the ACM SIGSOFT **Symposium on the Foundations of Software Engineering** (ESEC/FSE'2018), Orlando, Florida, USA. Topics include: architecture and design; components, services, and middleware; debugging; dependability, safety, and reliability; development tools and environments; distributed, parallel, and concurrent software; education; embedded and real-time software; empirical software engineering; formal methods, including languages, methods, and tools; model-driven software engineering; processes and workflows; program analysis; program comprehension and visualization; refactoring; reverse engineering; safety-critical systems; scientific computing; security and privacy; software economics and metrics; software evolution and maintenance; software modularity; software product lines; software reuse; testing; traceability; etc. Deadline for submissions: May 30, 2018 (journal-first papers), June 15, 2018 (new ideas and emerging results, industry papers, student research competition), June 25, 2018 (artifacts), June 29, 2018 (doctoral symposiums).

☺ November 05-06 ACM SIGAda's **High Integrity Language Technology International Workshop on Cyber-Security Interaction with High Integrity** (HILT'2018), Boston area, Massachusetts, USA. Co-located with SPLASH 2018. Organized by ACM SIGAda.

November 28-30 19th **International Conference on Product-Focused Software Process Improvement** (PROFES'2018), Wolfsburg, Germany. Topics include: experiences, ideas, innovations, as well as concerns related to professional software development and process improvement driven by product and service quality needs. Deadline for submissions: June 11, 2018 (full research and industry papers), August 5, 2018 (short papers, tools, demos, posters).

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

ptc® apexada | ptc® objectada®

Complete Ada Solutions for Complex Mission-Critical Systems

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

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

ptc

Ada-Europe 2018

23rd International Conference on Reliable Software Technologies
18-22 June 2018, Lisbon, Portugal

Advance Information

The 23rd International Conference on Reliable Software Technologies – Ada-Europe 2018 – will take place in Lisbon, Portugal. This conference is the latest in a series of annual international conferences started in the early 80's, under the auspices of, and organization by, Ada-Europe, the international organization that promotes the knowledge and use of Ada and Reliable Software in general into academia, research and industry.

Ada-Europe 2018 provides a unique opportunity for dialogue and collaboration between academics and industrial practitioners interesting in reliable software.

The conference will span a full week, including tutorials, workshops and a central three-day technical program with the latest advances in reliable software technologies and Ada. The core program features 3 keynote talks, 10 refereed scientific papers, 12 industrial presentations, 3 invited presentations and an industrial exhibition. The program is complemented with two workshops, on “Challenges and new Approaches for Dependable and Cyber-Physical Systems Engineering” (DeCPS) and on “Runtime Verification and Monitoring Technologies for Embedded Systems” (RUME). Half-day and full-day tutorials will be provided on Monday and Friday.

Week Overview

Monday	Tuesday	Wednesday	Thursday	Friday
5 Tutorials & RUME Workshop	**Keynote Talk** *Paulo Veríssimo*	**Keynote Talk** *Carl Brandon*	**Keynote Talk** *Erhard Plödereder*	5 Tutorials & DeCPS Workshop
	Regular session *Safety and Security*	**Regular session** *Handling Implicit Overhead*	**Industrial session** *V&V of Safety-Critical Software*	
	Industrial session *Ada in Industry*	**Industrial session** *Space Systems*	**Industrial session** *Software Methodologies*	
	Regular session *Ada 202X*	**Regular session** *Real-time Scheduling*	**Regular session** *New Application Domains*	
	Ada-Europe General Assembly	Conference Banquet & Best Paper Award	Best Presentation Award & Closing session	
	Welcome Reception			

http://www.ada-europe.org/conference2018

Keynote talks

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

- **Paulo Esteves-Veríssimo**, University of Luxembourg, Luxembourg, "*Security and Dependability Challenges of IT/OT Integration*"
- **Carl Brandon**, Vermont Technical College, USA, "*From Physicist to Rocket Scientist, and How to Make a CubeSat that Works*"
- **Erhard Plödereder**, University of Stuttgart, Germany, *"Vulnerabilities in Safety, Security, and Privacy"*

Tutorials

Bracketing the conference on Monday and Friday, the program includes ten tutorials:

- Recent Developments in SPARK 2014, Peter Chapin, Monday full day
- Access Types and Memory Management in Ada 2012, Jean-Pierre Rosen, Monday morning
- Design and Architecture Guidelines for Trustworthy Systems, William Bail, Monday morning
- Numerics for the Non-Numerical Analyst, Jean-Pierre Rosen, Monday afternoon
- Requirements Development for Safety- and Security-Critical Systems, William Bail, Monday afternoon
- Scheduling Analysis of AADL Architecture Models, Frank Singhoff and Pierre Dissaux, Friday full day
- Writing Contracts in Ada, Jacob Sparre Andersen, Friday morning
- Introduction to Libadalang, Raphaël Amiard and Pierre-Marie de Rodat, Friday morning
- Unit-Testing with Ahven, Jacob Sparre Andersen, Friday afternoon
- Frama-C, a Framework for Analysing C Code, Julien Signoles, Friday afternoon

Co-Located Workshops

The conference week features a new workshop, on the important topic of Runtime Verification and Monitoring Technologies for Embedded Systems (RUME), and the 5th edition of the International Workshop on Challenges and new Approaches for Dependable and Cyber-Physical Systems Engineering (DeCPS), following the success of the previous editions. The RUME workshop will take place on Monday, June 18th, and the DeCPS workshop will be on Friday, June 22nd, both from 09:30 to 17:30.

About the Venue

Lisbon is currently considered one of the best touristic cities in Europe. It is the capital of Portugal and is well known for its medieval castle, for the Belém tower, for its charming old neighbourhoods of Alfama and Bairro Alto, for the natural light and breath-taking scenery views, for the Fado music, for the sweet pastéis de Belém, and for so many other nice things that you should discover by yourself.

June is full of events in Lisbon, including the festivities in honour of St. António, with music, grilled sardines and popular parties in the old neighbourhoods. This year there is also the Rock in Rio Music Festival, starting on the weekend right after the conference. For all these reasons, you should book your hotel in advance! We arranged a block of rooms at the conference hotel with a special price for Ada-Europe (90€ for single and 100€ for double rooms, inc. breakfast), which can be reserved when registering for the conference. Room availability cannot be ensured after April 20.

The conference will take place at the VIP Executive Art's Hotel, in the modern Parque das Nações area. There you will find a large shopping mall, plenty of restaurants and bars, museums and the Lisbon oceanarium.

Social Events

The program includes 1-hour long coffee breaks, providing the opportunity for participants to discuss their work, to visit the exhibition and to socialise. Lunches will be served at the hotel restaurant, from Monday to Friday, providing further interaction opportunities. On Tuesday, participants are invited for a welcome reception after the Ada-Europe General Assembly, whose details are still being arranged. We hope to make everyone feel really welcome to Lisbon. And on Wednesday the day will end with the conference banquet, at the "Casa do Bacalhau" restaurant. The restaurant is located in the old stables of the Duke of Lafões palace. The room where dinner will be served has a wonderful ceiling, which is original from the 18th century. But most of all, we believe that you will fully enjoy the food and the wine!

Organization

General Chair: Nuno Neves (LASIGE/U. Lisboa, Portugal)
Program Chair: António Casimiro (LASIGE/U. Lisboa, Portugal)
Special Session Chair: Marcus Völp (U. Luxembourg, Luxembourg)
Tutorial and Workshop Chair: David Pereira (CISTER/ISEP, Portugal)
Industrial Co-Chairs: Marco Panunzio (Thales A.S., France) and José Rufino (LASIGE/U. Lisboa, Portugal)
Publication Chair: Pedro Ferreira (LASIGE/U. Lisboa, Portugal)
Exhibition Co-Chairs: José Neves (GMV Skysoft, Portugal) and Ahlan Marriott (White Elephant GmbH, Switzerland)
Publicity Chair: Dirk Craeynest (Ada-Belgium & KU Leuven, Belgium)
Local Secretariat: Madalena Almeida (Viagens Abreu, Portugal)

Program Committee

Mario Aldea (Universidad de Cantabria), Ezio Bartocci (Vienna University of Technology), Johann Blieberger (Vienna University of Technology), Rakesh Bobba (Oregon State University), Bernd Burgstaller (Yonsei University), António Casimiro (LASIGE/U. Lisboa), Juan A. de la Puente (Universidad Politécnica de Madrid), Virgil Gligor (Carnegie Mellon University), Michael González Harbour (Universidad de Cantabria), J. Javier Gutiérrez (Universidad de Cantabria), Jérôme Hugues (ISAE), Ruediger Kapitza (Technische Universität Braunschweig), Hubert Keller (Karlsruhe Institute of Technology), Raimund Kirner (Univ. of Hertfordshire), Adam Lackorzynski (TU Dresden and Kernkonzept GmbH), Kristina Lundkvist (Mälardalen University), Franco Mazzanti (ISTI-CNR), Laurent Pautet (Telecom ParisTech), Luís Miguel Pinho (CISTER/ISEP), Erhard Plödereder (Universität Stuttgart), Jorge Real (Universitat Politècnica de València), José Ruiz (AdaCore), Sergio Sáez (Universitat Politècnica de València), Elad Schiller (Chalmers University of Technology), Frank Singhoff (Université de Bretagne Occidentale), Jorge Sousa Pinto (University of Minho), Tucker Taft (AdaCore), Elena Troubitsyna (Åbo Akademi University), Santiago Urueña (GMV), Tullio Vardanega (Università di Padova), Marcus Völp (University of Luxembourg).

Industrial Committee

Ian Broster (Rapita Systems), Luís Correia (EMPORDEF-TI), Dirk Craeynest (Ada-Belgium & KU Leuven), Thomas Gruber (Austrian Institute Of Technology - AIT), Andreas Jung (European Space Agency), Ismael Lafoz (Airbus Defence and Space), Ahlan Marriott (White Elephant GmbH), Maurizio Martignano (Spazio IT), Marco Panunzio (Thales Alenia Space), Paul Parkinson (Wind River), Jean-Pierre Rosen (Adalog), José Rufino (LASIGE/U. Lisboa), Emilio Salazar (GMV), Helder Silva (EDISOFT), Jacob Sparre Andersen (JSA Consulting), Andreas Wortmann (OHB System).

Conference Sponsors (preliminary list)

AdaCore
RAPITA SYSTEMS LTD
CRITICAL SOFTWARE
Ciências ULisboa
LASIGE
PTC® Developer Tools
Ellidiss Software TNI Europe Limited
FCT Fundação para a Ciência e a Tecnologia

Additional support and sponsorship by:

Ada europe

In Cooperation with:

acm In-Cooperation
SIGAda
acm SIGBED
acm SIGPLAN
Ada Resource Association

http://www.ada-europe.org/conference2018

Call for papers and extended abstracts

HILT 2018

Workshop on *Languages and Tools for Ensuring Cyber-Resilience in Critical Software-Intensive Systems*

As part of SPLASH 2018, November 5 & 6, 2018, Boston, MA, USA
Sponsored by ACM SIGAda

The *High Integrity Language Technology* (HILT) 2018 Workshop is focused on the cyber-resilience needs of critical software systems, where such a system must be trusted to maintain a continual delivery of services, as well as ensuring safety in its operations. Such needs have common goals and shared strategies, tools, and techniques, recognizing the multiple interactions between security and safety.

We encourage papers and extended abstracts relating to:

- Language features that can be used to build security and/or safety into software-intensive systems;
- Approaches to apply effectively the emerging technologies of AI and Machine Learning in critical software systems;
- Mechanisms that can be used to understand, certify, and manage systems that are "data driven," relying on "soft code," where control flow and algorithms are expressed using data rather than "hard code" expressed directly in programming languages;
- Extending contract-based programming to specifying security resistance and resilience properties as well as safety and/or correctness properties;
- Strategies to minimize risk when applying complex software requirements to cyber-physical systems;
- Modeling and/or programming language features and analysis techniques that aid in code analysis and verification and that increase the level of abstraction and expressiveness;
- Language features that support continuous requirements maturation to support evolving needs, particularly in cyber-physical systems, while ensuring that security and safety properties are preserved.

This workshop is designed as a forum for communities of researchers and practitioners from academic, industrial, and governmental settings, to come together, share experiences, and forge partnerships focused on integrating and deploying tool and language combinations to address the challenges of building cyber-resilient software-intensive systems. The workshop will be a combination of presentations and panel discussions, with one or more invited speakers.

Attendees wishing to present at the workshop should prepare full papers (approx. 6-8 pages), or extended abstracts (approx. 2-4 pages) for their proposed presentations, and the workshop program committee will select presentations and organize them into sessions. Other interested participants are welcome to register for the HILT 2018 Workshop as part of their SPLASH 2018 registration.

July 31:	Papers or Extended abstracts due;
Aug 31:	Notification of submissions accepted for presentation
Sep 30:	Final submissions due
Nov 5&6:	Workshop as part of SPLASH 2018

Please submit papers and extended abstracts, by July 31, 2018, by following the link from: http://sigada.org/conf/hilt2018

Workshop Co-Chairs

- Bill Bail, MITRE
- Tucker Taft, AdaCore, Inc

Organizing Committee

- Dirk Craeynest, ACM SIGAda International Representative, KU Leuven
- Drew Hamilton, Chair, ACM SIGAda, Mississippi State University, CCI
- Clyde Roby, Secretary-Treasurer, ACM SIGAda, Institute for Defense Analyses
- Alok Srivastava, Editor, ACM Ada Letters, Engility Corp.
- Ricky E. Sward, Past Chair, ACM SIGAda, MITRE

URLs

- SPLASH 2018: http://www.splashcon.org
- HILT 2018: http://sigada.org/conf/hilt2018
- ACM SIGAda: http://sigada.org

Tools to get you there. Safely.
Ada and The GNAT Pro High-Integrity Family
www.adacore.com
AdaCore
The GNAT Pro Company

A New Approach Mining the SPL Feature Model and Design from Product Variants

J. Maâzoun, N. Bouassida
MIR@CL laboratory, Sfax University, Tunisia; email: jihenmaazoun@gmail.com, nadia.bouassida@isimsf.rnu.tn
H.Ben Abdallah
Abdulaziz University, Jeddah, Kingdom of Saudi Arabia; email: hbenabdallah@kau.edu.sa

Abstract

Feature models are a popular means to express requirements of software product lines in a given domain at an abstract level. They are used to describe variable and common properties of products in a product line, and to derive and validate configurations of software systems. Existing feature model identification methods start from a set of product variants to extract the common and variable features. They rely on the hypothesis that the product variants have the same structure and names, which may not be always the case. In this paper, we present a method that lifts this restricting hypothesis by using text mining techniques. We empirically compare the quality of the feature models produced by our method to those constructed by experts.

Keywords: Software product line, SPL, feature model, SPL design.

1 Introduction

A Software Product Line (SPL) [1] represents a family of systems in a given domain, that share a group of manageable features each of which is seen as an end-user, visible characteristic of the system [2]. It provides for predictive and organized software reuse. Generally, an SPL is modeled through a feature model (or diagram) that specifies the common features and variation points among the systems in the SPL domain.

A feature model can be constructed either in a bottom-up (cf., [3,4,5]) or top-down (cf., [6]) approach. A top-down development approach starts with a domain analysis to construct the feature model of an SPL; that is, this approach is driven by the functional requirements towards the definition of alternative solutions. It is best applied when the domain has not yet been sufficiently explored. However, this approach is time consuming and requires guidelines (so far undefined) for the domain requirements analysis. On the other hand, a bottom-up approach starts from the code of a set of existing products in a given domain and it identifies their common and variable features.

Most proposed bottom-up feature model construction methods examine a set of product variants code to identify common blocks of elements each of which represents a feature. They first use textual similarity measures to identify the source code elements that constitute a feature. Then, based on these measures, they extract features and/or feature models using either information retrieval techniques (cf. [5]) or genetic algorithms (cf. [7]) by clustering elements with "similar" names. In this identification process, they adopt different similarity measures and clustering techniques. Consequently, they differ in the granularity and cohesiveness of the identified features, which in turn affects the structure of the constructed feature model. Evidently, this difference also impacts the reuse of the SPL when deriving a new product.

In recent years, several researchers examined the extraction of features and/or feature models from product source codes. Existing feature identification approaches [3,4,5,7,8] rely on two essential hypotheses: all source codes use the same vocabulary to name packages, classes, attributes and methods; and the product variants have very similar/identical structures. These assumptions stem from their way of seeing how the product variants were created: essentially through “copy-and-paste” operations which, indeed, preserve the names and cause little structuring changes. However, these approaches cannot be applied in the general setting where an SPL should be constructed from product variants that were produced by different developers, and/or product variants that endured so many modifications that the same names and structure assumptions are violated. For instance, a class in one product can be represented in a second product through two classes where the attributes and methods of the original class are distributed. A second example of product variability is when a class in one product was moved from one package to another package. In this case, we treat each element independently of its possessing class. For these simple examples, existing feature identification approaches would fail. In fact, unlike Al-Msiedeeen et al. [5], we do not consider the owner of each element when constructing both FM and design. Furthermore, we can find two methods having the same name but different body or two methodes having different name but the same body. To verify this variability, we use a code-clone detection technique.

The herein presented bottom-up feature model extraction method has four main merits. First, it accounts for the differences in the structures and element names of the product variants. Secondly, after a comparative evaluation of three popular textual similarity measures (LSI, PCA and TF-IDF), our method adopts LSI which has the highest precision in feature identification. Third, our method extracts the SPL design along with its feature model. For this purpose, we use reverse engineering techniques to extract the design (class and sequence diagrams) of the product variants. Then, we use the the Density Based Spatial Clustering of Applications with Noise (DBSCAN) technique [9] to unify the design of the different products and to extract the design of the SPL. At the same time, we use the FCA [10] and LSI (Latent Semantic Indexing) [11] and DBSCAN clustering techniques to extract the feature model from the source code of the product variants. The design of the SPL is enriched with the information contained in the feature model. Fourth, our method is supported by a tool set named "SPL-Design" that automates its steps.

The remainder of this paper is organized as follows. Section 2 overviews feature model and existing works interested in the extraction of SPL from products source code. Section 3 presents our technique for feature model extraction. Section 4 presents our technique for SPL design extraction from product source codes. Section 5 presents our tool named "SPL-Design" which is used to evaluate empirically the quality of feature models produced by our method. Finally, Section 6 summarizes the paper and outlines future work.

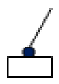
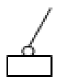
2 Related work

We propose in this section a brief overview of feature models and bottom up processes to extract feature models.

A feature model is a hierarchical model that captures the commonality and variability of a product line. It is used to derive and validate configurations of software systems. As introduced by the FODA method [2, 12], feature models have a tree structure, with features forming nodes of the tree. Feature variability is represented by the arcs and groupings of features. Relationships between a parent (or compound) feature and its child features (or subfeatures) designate the following selection strategies among features when deriving a product in the line of the SPL.

Feature models have also been used to improve program comprehension. In most re-engineering activities, the source code is the only reliable source of information. Feature models allow designers to bridge the gap between the concrete code and the fairly abstract information of documents such as architecture design. They are applied in the approaches supporting reverse engineering in a hypothesis-verification procedure [13].

A Feature model has a tree structure where each node represents a feature. Feature variability is represented by the arcs and groupings of features. There are two different types of feature groups:

- **Mandatory**  : Child feature is obligatory
- **Optional**  : Child feature is optional

In addition to the parental relationships between features, cross-tree constraints are allowed. Five common cross-tree constraints are:

- **And** connected (sub)features must all be selected together when deriving a new product from the SPL.
- **Xor** connected (sub)features indicate that only one feature can be selected when deriving a new product from the SPL.
- **Or** connected (sub)features indicate that one or more (sub)feature(s) can be selected when deriving a new product from the SPL.
- **Require** indicates that the selection of one (sub)feature necessitates the selection of the other.
- **Exclude** indicates that two (sub)features cannot be part of the same product derived from the SPL.

Note that a feature can be either simple/elementary like a package and a class, or composed of several elements like (package, Class), (package, Class, attribute, method)...

Several works investigated feature model extraction from the source code of products in order to construct the SPL ([3, 5, 13, 14]). For instance, Ziadi et al., [3, 15] propose an approach that first abstracts the input products in SoCPs (Sets of Construction Primitives) and, secondly, it identifies features by determining common and intersecting SoCPs. This approach was validated using two case studies: a banking example and the Argo-UML software product line [16]. The obtained results show that the approach can handle products with variable names for classes, methods and attributes. However, this approach produces a feature model which contains only one mandatory feature and the others are considered as optional features. Thus, it identifies neither separated mandatory features, nor alternative features and their related constraints such as the mutual exclusion.

Al-Msiedeeen [5] propose an approach based on the definition of the mapping model between OO elements and feature model elements. This approach uses FCA to cluster similar OO elements into one feature. It uses LSI to define a similarity measure based on which the clustering is decided. This approach improves the approach of Ziadi [3] since it extracts mandatory features and optional features along with some constraints among features like "And" and "Require". However, it does not treat product variants with different structures or different terminologies. Moreover, Al-Msiedeeen et al. ([5, 17, 18]) use, in the step of extracting features FCA and LSI, while the input of FCA 0 or 1 and the results of LSI are real values. Finally, it does not extract the design which facilitates the SPL comprehension.

Salman et al., [7, 19] present a genetic algorithm to recover traceability links between feature models and source code. Traceability links in SPL are needed to relate variation points and variants with all corresponding low level artifacts (requirements, design, source code and test cases artifacts). The genetic algorithm can determine approximately the implementation of each feature (by linking the feature to classes). However, it generates just one solution for each run, and the number of runs necessary to determine all possible classes for each feature is unknown.

Note that all source code in the approaches of [3, 5, 7] use the same vocabulary and the same structure. However, these approaches cannot be applied in the derivation of an SPL from product variants that were produced by different developers. Moreover, in case of maintenance or evolution, all feature models extracted through these works do not produce information at an intermediate level of abstraction between the code and the feature model. Such information is vital for comprehension, maintenance and evolution purposes. Without it, maintainers and developers end up spending a lot of time and effort to understand the behavior and structure of the system. To provide for these purposes, we propose to have a design that accompanies the feature model and the source code of the SPL.

3 Feature mining process

In this section, we present a bottom-up process that extracts from the source code of product variants, the feature model. Our approach contains four steps: Name harmonization, commonalities and variability identification and feature model extraction (see figure 1).

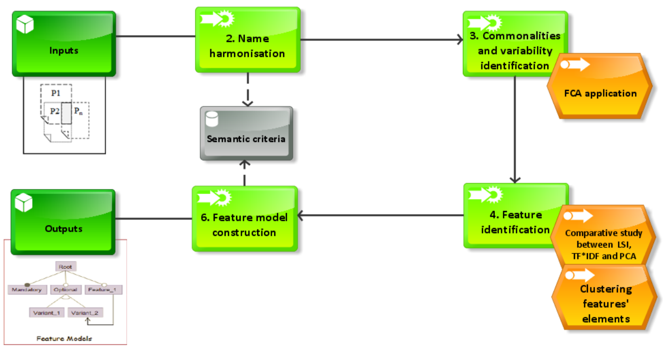

Inputs
P1
P2
Pn
2. Name harmonisation
3. Commonalities and variability identification
FCA application
Semantic criteria
Outputs
Root
Mandatory
Optional
Feature_1
Variant_1
Variant_2
Feature Models
6. Feature model construction
4. Feature identification
Comparative study between LSI, TF*IDF and PCA
Clustering features' elements

Figure 1: Feature mining process

3.1 Name Harmonization

This pre-processing step starts by identifying the semantic correspondences between the names of packages, classes, methods and attributes names through interrogating WordNet [20]. The semantic relations are examined in the following order: the equivalence (Synonyms), the string extension (str_extension), and then the generalization (Hypernyms) [21].

In order to determine the correspondences between the names, we propose a set of semantic criteria. The following five criterias express linguistic relationships between element names (however, the list can be extended):

- Synonyms(C1,· · · ,Cn): implies that the names are either identical or synonym, *e.g.*, Mobile-Mobile and Phone-Mobile.
- Hypernyms(C1; C2,· · · ,Cn): implies the name C1 is a generalization of the specific names C2, · · · , Cn, *e.g.*, Media-Video.
- str_extension(C1; C2): implies that the name C1 is a string extension of the name of the class C2, *e.g.*, Image-NameImage.
- Hyponyms(C1; C2):implies that the name C1 is included in the meaning of another C2 more general word, *e.g.*,Audio-Media.
- Antonyms(C1; C2): implies that the name C1 is opposite in meaning to another C2, *e.g.*, full-empty.

The determination of the above linguistic/semantic relationships can be handled through either a dictionary (*e.g.*, Wordnet), or a domain ontology when available.

The above semantic criteria are insufficient when two methods have the same or synonymous names but different bodies. Then, it is necessary to verify the variability between methods' body. To resolve this problem and to verify the variability encapsulated in the body of the methods, we adopt the code cloning technique presented in [22].

As presented in Figure 2, we remark that in Wordnet dictionay, "Search" and "Explore" are synonyms and have the same bodies, consequently, we harmonize their names. Note that, if the bodies of both methods were detected as different by our cloning algorithm, we wouldn't have harmonized their names.

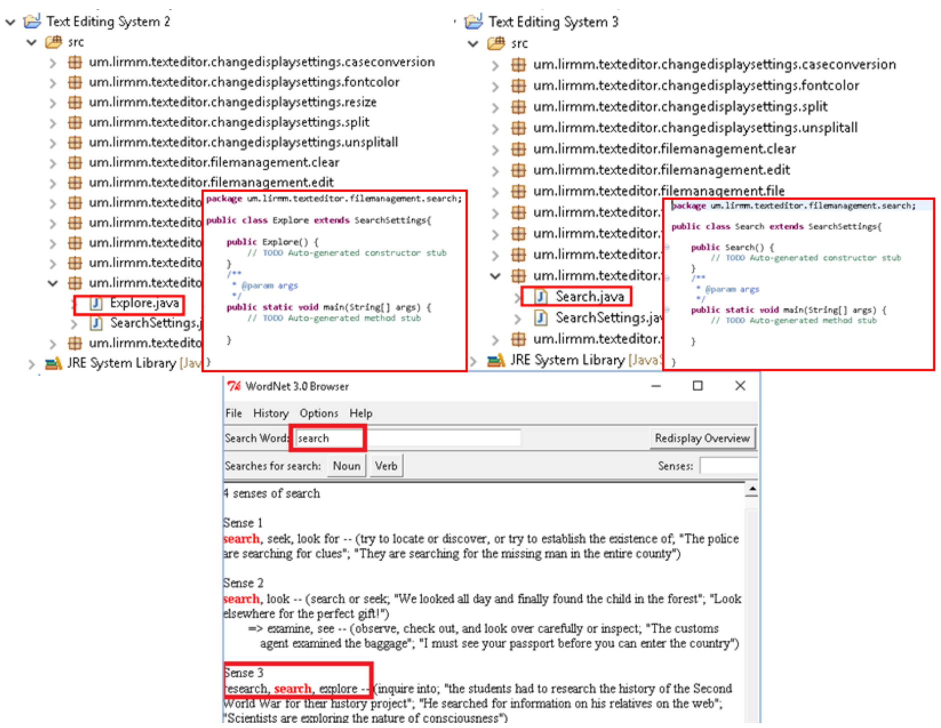
At the end of the pre-processing step, all semantically related names would be harmonized and can then be analyzed through the FCA in the features identification step.

3.2 Commonalities and variability identification

In this step, we use Formal Concept Analysis (FCA) to extract the commonalities and variability among the harmonized product variants. Before explaining this step, let us first overview the basics of FCA. FCA [10] is a method of data analysis with a growing popularity across various domains. The main idea of FCA is to analyze data described through the relationships among a particular set of data elements. In our approach, the data represent the product variants being analyzed; the data description is represented through a table where the product variants constitute the rows while source code elements (packages, classes, methods, attributes) constitute the columns of the table. Due to space limitations, Figure 3 shows an extract of this table.

As illustrated in the table of Figure 3, certain elements are common blocks which are commonly used in all products like Package(ChangeDisplaySetting). Other elements appear in specific products. For example, Class(Replace), Class(ReplaceAll), Class(SearchSetting), Class(Search), Method(SearchSetting) and Method(Search) appear in product 1,3,5 and 7. Then, these elements belong to the same block of variations.

From this table, a concept lattice is derived. It allows first to define commonalities and variations among all products. The top element of the lattice indicates that certain objects have elements in common (*i.e.*, Package(ChangeDisplaySetting) presented in the table of figure 3), while the bottom element of the lattice show that certain attributes fit common objects



```
um.lirmm.texteditor.changedisplaysettings.caseconversion
um.lirmm.texteditor.changedisplaysettings.fontcolor
um.lirmm.texteditor.changedisplaysettings.resize
um.lirmm.texteditor.changedisplaysettings.split
um.lirmm.texteditor.changedisplaysettings.unsplitall
um.lirmm.texteditor.filemanagement.clear
um.lirmm.texteditor.filemanagement.edit
um.lirmm.texteditor.filemanagement.file
Explore.java
SearchSettings.java
Search.java
```

```
package um.lirmm.texteditor.filemanagement.search;

public class Explore extends SearchSettings{

    public Explore() {
        // TODO Auto-generated constructor stub
    }
    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }
}
```

```
package um.lirmm.texteditor.filemanagement.search;

public class Search extends SearchSettings{

    public Search() {
        // TODO Auto-generated constructor stub
    }
    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }
}
```

Search Word: search

4 senses of search

Sense 1
search, seek, look for -- (try to locate or discover, or try to establish the existence of; "The police are searching for clues"; "They are searching for the missing man in the entire county")

Sense 2
search, look -- (search or seek; "We looked all day and finally found the child in the forest"; "Look elsewhere for the perfect gift!")
=> examine, see -- (observe, check out, and look over carefully or inspect; "The customs agent examined the baggage"; "I must see your passport before you can enter the country")

Sense 3
research, search, explore -- (inquire into; "the students had to research the history of the Second World War for their history project"; "He searched for information on his relatives on the web"; "Scientists are exploring the nature of consciousness")

Figure 2: An example of name harmonization step

	Package(Changedisplay setting)	Class(uppercase)	Class(split)	Class(splitVertical)	Class(splitHorizental)	Class(Replace)	Class(ReplaceAll)	Class(SearchSetting)	Class(Search)	Method(SearchSetting)	Method (Search)	Method(uppercase)	Method(SetUppercase)
TextEditingSystem1	X	X	X	X	X	X	X	X	X	X	X	X	X
TextEditingSystem2	X		X	X	X								
TextEditingSystem3	X	X				X	X	X	X	X	X	X	X
TextEditingSystem4	X	X										X	X
TextEditingSystem5	X	X				X	X	X	X	X	X	X	X
TextEditingSystem6	X	X	X	X	X							X	X
TextEditingSystem7	X	X				X	X	X	X	X	X	X	X
TextEditingSystem8	X	X	X	X	X							X	X

Figure 3: Part of the formal context describing TextEditing systems by source code elements

(variations). In our example,in the table of figure 3, we remark that the elements Class(Replace), Class(ReplaceAll), Class(SearchSetting), Class(Search) appearing in specific products, belong to the same block of variations in figure 4. We note that elements Common blocks and blocks of variation are composed of atomic blocks of variation representing only one feature.

3.3 Feature identification

The objective of this work is to analyze the effects of different information retrieval techniques on the feature identification step. As explained in the previous section, this step starts after the common blocks and the variable elements have been identified. It then applies one of the existing textual similarity techniques (LSI, TF-IDF, or PCA).

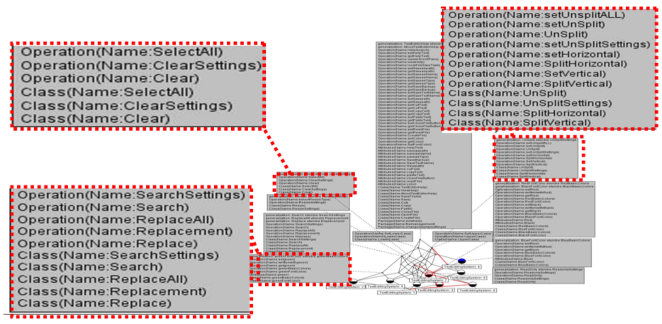

Operation(Name:SelectAll)
Operation(Name:ClearSettings)
Operation(Name:Clear)
Class(Name:SelectAll)
Class(Name:ClearSettings)
Class(Name:Clear)
Operation(Name:setUnsplitALL)
Operation(Name:setUnSplit)
Operation(Name:UnSplit)
Operation(Name:setUnSplitSettings)
Operation(Name:setHorizontal)
Operation(Name:SplitHorizontal)
Operation(Name:SetVertical)
Operation(Name:SplitVertical)
Class(Name:UnSplit)
Class(Name:UnSplitSettings)
Class(Name:SplitHorizontal)
Class(Name:SplitVertical)
Operation(Name:SearchSettings)
Operation(Name:Search)
Operation(Name:ReplaceAll)
Operation(Name:Replacement)
Operation(Name:Replace)
Class(Name:SearchSettings)
Class(Name:Search)
Class(Name:ReplaceAll)
Class(Name:Replacement)
Class(Name:Replace)

Figure 4: The lattice for the formal context

3.3.1 *LSI application:*

LSI (Latent Semantic Indexing) [23] assumes that words that always appear together are related [11] and that words that are used in the same contexts tend to have similar meanings. Consequently,it would be beneficial to use it in identifying features based on their textual similarity. Similarity between lines is described by a similarity matrix where the columns and rows represent lines vectors. Thereafter, a document-term matrix (TDM) and a matrix-term query is constructed for each line in the block to save the variation occurrences of terms in a single collection of document.

LSI uses each line in the block of variations as a query to retrieve all lines similar to it, according to a cosine similarity. In our work, we consider the most widely used threshold for cosine similarity that equals 0.70 [11].

3.3.2 *TF-IDF application:*

TF-IDF (Term Frequency-Inverse Document Frequency) [24] uses term weighting to determine the importance of a term in a document.

The computation of terms weights is followed by the calculation of a similarity measure which is the cosine, as follows:

$$sim(di, q) \approx \cos(\overrightarrow{di}, \overrightarrow{q}) = \frac{\sum_{tj \in T} W_{ij} W_{qj}}{\sqrt{\sum_{tj \in T} W_{ij}^2 W_{qj}^2}} \quad (1)$$

where: d_i is the document i; q is the query; W_{ij} is the weight of the term t_j in d_i;W_{qj} is the weight of the term t_j in q; and T is the set of terms contained in the documents.

We use the TF-IDF method to count the frequency of terms in all the documents. The document contains names of elements (package, class, method, attribute) in a block.

3.3.3 *PCA application:*

Principal component analysis (PCA) [25] is a statistical technique for information extraction. It identifies patterns in data and expresses the data in such a way as to highlight their similarities and differences [26].

First, PCA identifies a new set of orthogonal coordinate axis by finding the direction of maximal variance through the coordinates. Once the first principal component has been obtained, we can use orthogonal projection to map the coordinates down onto this new axis.

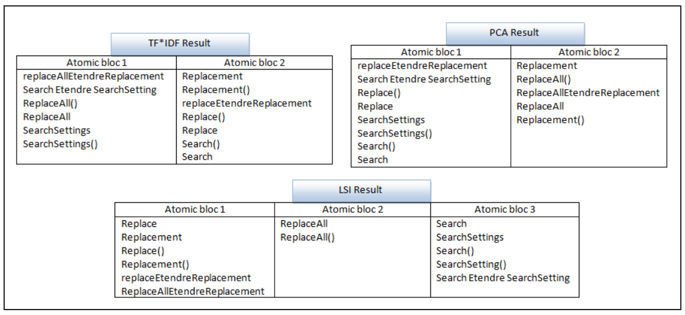
The PCA then calculates the second greatest variance on the the second coordinate called second principal coordinate (axis) which is both orthogonal to the first principal component, and is the next best direction for approximating the original data. Then, PCA calculates the third greatest variance and so on.

3.3.4 *Clustering of features' elements:*

To cluster elements and identify features, Msie'ddine et al. [5] use LSI and FCA to identify features. However, FCA use as input 0 or 1, while the result of LSI are real values. To solve this problem, we use another clustering technique (DBSCAN) [9] which is a topometric algorithm used to cluster spatial data. This algorithm was chosen since it does not limit the number of clusters. we propose to use the Density Based Spatial Clustering of Applications with Noise (DBSCAN). It is a topometric algorithm used to cluster spatial data. This algorithm was chosen since it does not limit the number of clusters. We used Weka (Waikato Environment for Knowledge Analysis) for the implementation of DBSCAN algorithm. Weka is a popular suite of machine learning software written in Java, developed at the University of Waikato, New Zealand [27].

The different matrices resulting for the LSI, PCA, TF-IDF are used as input for the DBSCAN to group the similar elements together based on the lexical similarity. The result of LSI is presented through three clusters. However, only two clusters are identified after the use of PCA and TF-IDF techniques.

The resulting atomic blocks (features) are displayed in Figure 5.



TF*IDF Result

Atomic bloc 1	Atomic bloc 2
replaceAllEtendreReplacement	Replacement
Search Etendre SearchSetting	Replacement()
ReplaceAll()	replaceEtendreReplacement
ReplaceAll	Replace()
SearchSettings	Replace
SearchSettings()	Search()
	Search

PCA Result

Atomic bloc 1	Atomic bloc 2
replaceEtendreReplacement	Replacement
Search Etendre SearchSetting	ReplaceAll()
Replace()	ReplaceAllEtendreReplacement
Replace	ReplaceAll
SearchSettings	Replacement()
SearchSettings()	
Search()	
Search	

LSI Result

Atomic bloc 1	Atomic bloc 2	Atomic bloc 3
Replace	ReplaceAll	Search
Replacement	ReplaceAll()	SearchSettings
Replace()		Search()
Replacement()		SearchSetting()
replaceEtendreReplacement		Search Etendre SearchSetting
ReplaceAllEtendreReplacement		

Figure 5: The resulting features extracted with the different textual similarity measures

To evaluate the best textual similarity measure for feature extraction, we used two clustering evaluation measures, precision and recall, which are defined as follows:

$$Recall_{(i,j)} = \frac{n_{ij}}{n_i}$$

$$Precision_{(i,j)} = \frac{n_{ij}}{n_j}$$

where n_{ij} is the number of elements present in the correct or true atomic block obtained by our tool (feature) F_i and in the resulting atomic block (feature) F_j. n_i represent the total number of elements in correct or true feature F_i. n_j represent the total number of elements in the block F_j obtained by our method.

For every feature model in our evaluation corpus, we selected ten blocks and we apply the different textual similarity measures for feature extraction. Table 1 shows a comparison of the recall and precision values of the different techniques. After calculating the average of recall and precision for all features, we remark that the LSI technique is the most suitable for features extraction since it has the best recall which equals to 0.877 and the best precision which equals to 0.686.

A main advantage of LSI is its ability to generate more efficent atomic blocks. In addition, LSI is capable of assuring decent results, much better than PCA and TF-IDF. LSI is performing better since it appear together are in the same context.

As a conclusion, in our approach, we use LSI to extract both feature model and SPL design. The next step in our approach determines the hierarchy and constraints among features and finalizes the feature model construction.

3.4 Feature model construction

This phase has a threefold motivation. First, the features which are composed of many elements (packages, classes, attributes, methods) are renamed based on the frequency of the names of its elements. In addition, the organization and structure of the features is also retrieved based on the semantic criteria. In fact, since the owner information was omitted, then to retrieve the organization of the features, we use the semantic criterion and concept lattice.

Table 1: Recall and precision calculus

	FM1		FM2		FM3		Average	
Technique	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision
LSI	0.9	0.85	0.857	0.46	0.904	0.75	0.887	0.686
PCA	0.7	0.75	0.625	0.38	0.5	0.33	0.608	0.48
TF-IDF	0.6	0.28	0.6	0.3	0.66	0.21	0.62	0.26

In the first time, the concept lattice permits to define commonalities and variations among all products. The top element of the lattice indicates that certain objects have elements in common, while the bottom element of the lattice show that certain attributes fit common objects (variations).

To deduce the hierarchy of features, we use the semantic criterion *hypernym* and *hyponym*. The extraction of *hypernym* and *hyponym* can be done through the WordNet. In our case, if we have two features, F1 and F2; if F1 is an hypernymy or hyponym of F2 then F1 is the parent of F2. For example, considering the features: "File management", "read only", we note that the word "File" is an hypornym of "read"; consequently, the feature "File management" is the parent of the feature "read only".

After deducing the hierarchy of features, it is necessary to define the constraints between them. For this purpose, we apply some rules based on semantic criterion and the concept lattice. Consequently, the lattice indicates the relationships among features. The following relationships can be automatically derived from the sparse representation of the lattice and presented to the analyst:

- *Mandatory*: The features appearing at the top concept in the lattice used in every product.
- *Optional*: The features appearing at the bottom concept in the lattice used in some product.
- *Or*: Can be deduce from two facts:
  - *Antonym relation:* If we have two features, F1 and F2; if F1 is an *Antonym* with F2 then F1 has an *Or* constraint with F2. For example, the feature "Open" has an *antonymy* relation with "Close". Then, the relation between them is "Or".
  - *str_extension relation:* If a feature F1 is an *str_extension* with a feature F2 then F1 has an *Or* constraint with F2.
- *Xor*:
  - If two features F1 and F2, having the same parent, that never appear together in FCA cross-table and always one of them is present then F1 has an *Xor* constraint with F2.
  - *synonym relation*: If a feature F1 is an *synonym* with a feature F2 then F1 has an *Xor* constraint with F2.
- *Exclude*: If two features F1 and F2, having different parents, that never appear together in FCA cross-table and always one of them is present, then F1 has an *exclude* constraint with F2.
- *Require*: Can be deduce from two facts:
  - If the appearing of feature F1 requires the appearing of feature F2, then F1 has an *Require* constraint with F2.
  - If an element (package, class) in one feature F1 have elements (Method, attribute) belonging in a second feature F2, then F1 has a *require* constraint with F2.
- *AND*: Two features F1 and F2 that appear in the same concept

At the end of this last step, all the features are collected in a feature model to specify the variations between these products.

4 SPL Design mining process

In this section, we present a bottom-up process that extracts from the source code of product variants, the SPL design enriched with information extracted from the feature model. We adopt the same step as presented in feature model mining process and we add the step of reverse enginnering. Next, all steps of design elements extraction and SPL design construction will be applied to the design. For the construction of the SPL with some differences since we work on the design, we use our UML profile named UML-SPL presented in Maazoun et al. [28] to resolve the problem of traceability between design and feature model. Our UML profile enriches the UML diagrams with informations extracted from the feature model and highlights the variability of the SPL. It treats the static and dynamic aspect. It defines the concept of recommendation at class and sequence diagrams. It integrates also OCL(Object-Constraint Language) constraints ensuring the consistency of the variation points.

4.1 Reverse engineering

Following Name harmonization, we needed to reverse engineer the code to construct the class and sequence diagrams required in the feature extraction step of our process. A class diagram contains all of the classes and enumerates the relationship between them (association, inheritance, composition, aggregation...). A sequence diagram contains Lifelines, message, operation, object...

In order to construct the class and sequence diagrams, we reverse engineered the code using plugin eUML [1] for eclipse [2].

[1]http://www.soyatec.com/euml2/installation/
[2]https://eclipse.org/downloads/

4.2 Design elements extraction

In order to tolerate some difference among the design of the product variants, we adapt the FCA [11]. In this phase, the FCA is applied to extract the common elements and the variable elements of the design. The data description is represented through a table where the product variants constitute the rows while class' diagram elements (packages, classes, methods, attributes)and relationship between classes constitute the columns of the table.

Then, a concept lattice is derived.The top element of the lattice indicates the common elements while the bottom element of the lattice show variations of certain attributes. This process permits us, first, to derive design enriched with optional and mandatory.

The organization and structure of the SPL design is retrieved based on the construct rules defined in the next section.

4.3 SPL design construction and enrichement with Feature Model

To construct and organize our SPL design, we define some rules which used our proposed UML profile "SPL-UML". These rules are:

- **R1:** Each mandatory class will be presented with its mandatory elements (attribute, method).
- **R2:** If a relationship is mandatory, then startAssociation and the endAssociation are mandatory and it will be present in the design.
- **R3:**If a relationship has a startAssociation or an endAssociation mandatory, will be present in the design and the optional startAssociation or endAssociation will be present and stereotyped "recommended".
- **R4:** The rest of the optional elements will be present in the design according to the degree of its presence in all the class' diagrams.

Finally, we propose to represent the design of the SPL using our UML profile enriched with the information extracted from the generated feature model.

Consequently a generic class' diagram is derived. This diagram is enriched by information extracted from the feature model illustrated in figure 6.

In our running example, we present the SPL design construction. In fact, by applying the rule R1, all mandatory elements will be stereotyped with "mandatory" and "feature_name" and the relation between them must be mandatory and presented with a bold line. For example, classes "File", "Text" are mandatory and the relation between them are also mandatory by applying the rule R2.

5 Evaluation

The overall objective of this section is to show the ability of our method and tool to evaluate feature model, SPL design extracted from product variants having different structure and naming.

5.1 SPL Design tool

SPL Design tool automates all the steps of the feature and SPL design mining process.

In the first step, the user chooses the source code file, then the tree will be extracted and saved in an XMI document. The XMI document corresponds to the name of elements of the parsed code (package, class, method, attribute)(see interface 1 of figure 7).

After applying the name harmonization, the XMI file will be updated (see the interface 2 of figure 7) and will be an input of FCA method (see the interface 3 of figure 7) and the concept lattice will be derived (shown in the interface 4 of figure 7). Then, common blocks and blocks of variation will be determined (shown in the interface 5 of figure 7).

Common blocks and blocks of variations are composed of atomic blocks of variation representing only one feature. To define features, we apply LSI with Matlab by clicking to the button "Apply LSI". According to a cosine similarity that is equal to 0.70, LSI uses each line in the block of variations as a query to retrieve all lines similar to it. The similarity matrix which is the LSI result is used as input for the DBSCAN to group the similar elements together based on the lexical similarity.

5.2 Feature model and SPL design evaluation

This section aims at evaluating both the feature model and the SPL design extracted from product variants having different structure and naming and generated by our tool. For this purpose, we have developed in our team different products in the domain of Mobile Media, games (Sodoku, Tankwar, Acrade Maker),DB system. For every SPL, we select five products. The products were developed by different persons and consequently the naming and structure were different. In every product, the number of package do not exceed five packages, the number of classes do not exceed 27 classes. Then, we applied our approach to obtain the SPL design and feature model. Afterwards we compared the obtained SPL design and feature model with an existing FM and the SPL design from FeatureHouse [3]. The design of the SPLs extracted from FeatureHouse was reverse engineered with le plagin e-UML for eclipse.

Our evaluation starts by comparing the feature models obtained by our tool vs. feature models handled by experts. This comparison focused namely on the different elements (i.e. features, packages, classes, methods, attributes) and constraints that compose the feature model. More specifically, we used the recall and precision measures. Similarly, we relied on a comparative evaluation of the design generated by our tool vs. the design produced by experts, based on these measurements. The precision represents the ratio of the number of correct elements and constraints (respectively the design elements) detected by our tool among the total number of the generated elements and constraints (respectively the design elements), whereas the recall represents the ratio number of correct elements and constraints(respectively the

[3]http://www.infosun.fim.uni-passau.de/spl/apel/fh/

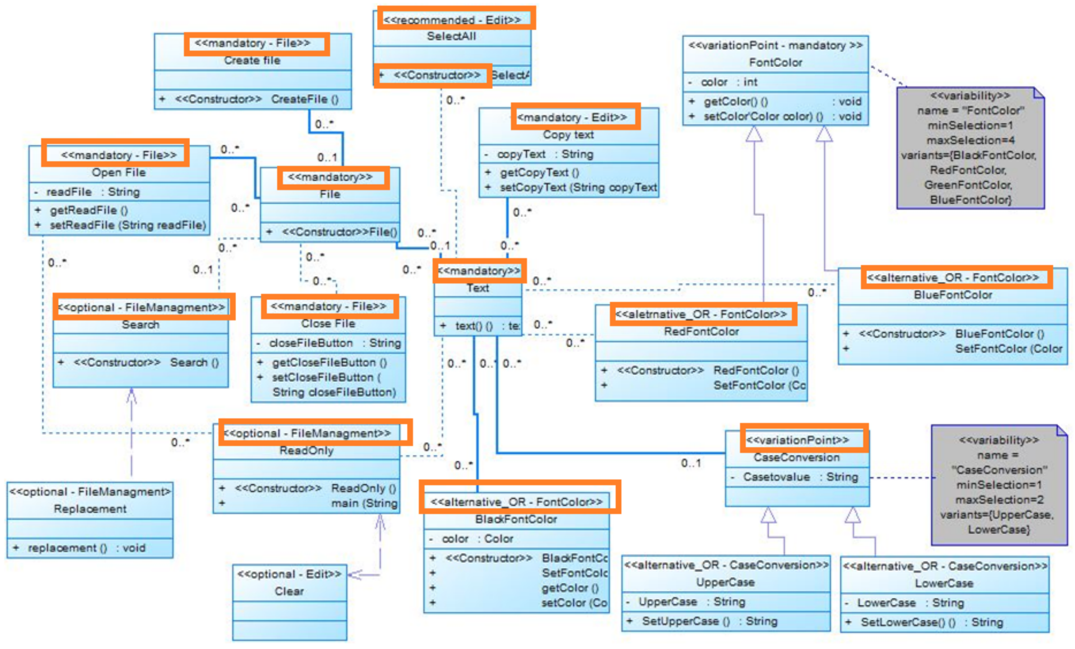


Figure 6: The class diagram of the SPL represented with SPL-UML.

Table 2: Evaluation results

Evaluation	TP	FP	FN	Precision= TP/(TP+FP)	Recall= TP/(TP+FN)
Feature model	46	16	5	0.74	0.90
SPL design	56	15	4	0.78	0.93

design elements) belonging to the feature model generated by our tool and the total number of the elements and constraints (respectively the design elements) handled by experts. For this reason, we count the number of True Positives (TP), False Positives (FP), and False Negatives (FN). False Positives are elements and constraints belonging to the feature model (or design) wrongly identified. False Negatives are elements belonging to the feature model (or design) identified by expert that our tool could not generate.

In our feature model evaluation (Table 2), the average precision of 0.74, is explained by the fact that we found some false positive features (i.e. incorrect detected features, incorrect detected constraints). Compared to the true positives found by our method, the false positives elements are not significant. The recall, whose average value is 0.90, indicates that we have also some false negative features and some false negative constraints (i.e. true features not detected, false constraints).

In our SPL design evaluation (Table 2), the average precision of 0.78, is explained by the fact that we found some false positive classes (i.e. incorrect detected classes or methods belonging to incorrect class). The recall, whose average value is 0.93, indicates that we have some false negative classes (i.e. true associations between classes not detected).

Threats to validity. An important concern of our approach is the rely on the WordNet ontology; in fact, this latter cannot recognize the semantic relations between all the words that it contains. This fact causes the inability of our approach to generate some features and constraints. For instance, in the context of mobile media domain, the product variants contain, in particular, two different methods: "*RemoveAlbum*" and "*DeleteAlbum*". The problem is that the WordNet does not consider the word "*remove*" as a synonym of "*delete*". Thus, the name harmonization step would not be performed by our approach, which leads to the generation of incorrect feature. In addition, in the reverse engineering step of our SPL design mining process, some relations (aggregation or composition) could not be detected; this fact is caused by the plugin e-UML for Eclipse.

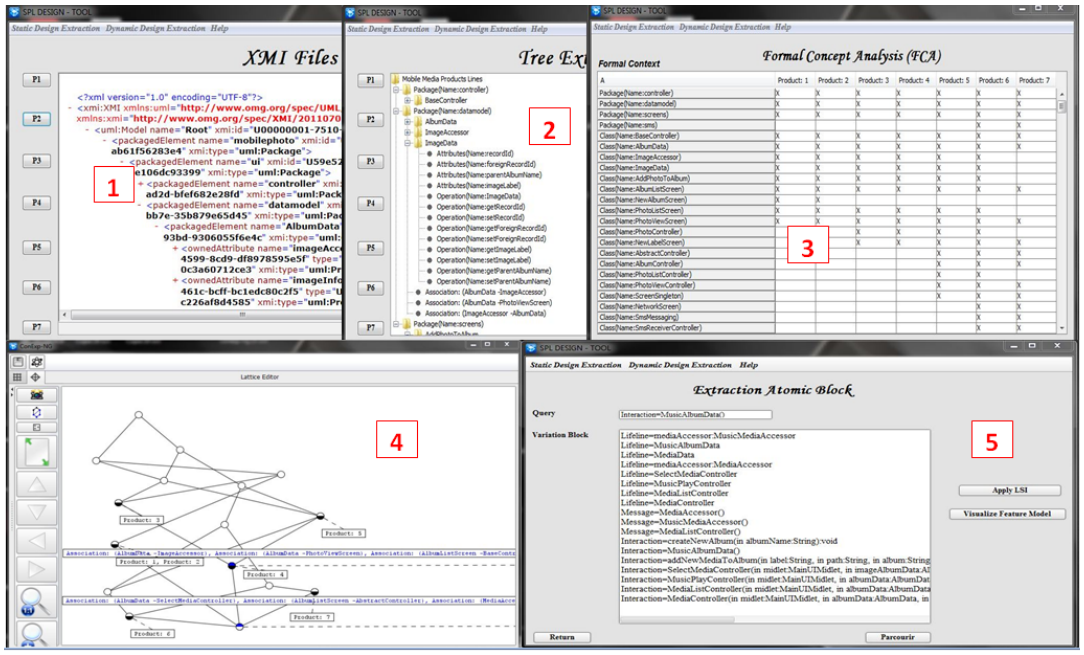
5.3 Feature model quality evaluation

The overall objective of this section is to show the ability of our method and tool to generate a feature model with a quality similar to the quality of existing SPLs in the same domain. For this purpose, we evaluated our method through a quantitative, empirical evaluation based on a comparison between our feature models and feature models constructed by experts.

More specifically, our empirical study took five feature models existing in FeatureHouse [4]:

- FM1: Feature model for TankWar game.
- FM2: Feature model for MobileMedia system.
- FM3: Feature model for BerkeleyDB system.
- FM4: Feature model for Sodoku game.
- FM5: Feature model for Acrade Game Maker.

[4]http://www.infosun.fim.uni-passau.de/spl/apel/fh/



```
<?xml version="1.0" encoding="UTF-8"?>
- <xmi:XMI xmlns:uml="http://www.omg.org/spec/UML
  xmlns:xmi="http://www.omg.org/spec/XMI/2011070
  - <uml:Model name="Root" xmi:id="U00000001-7510-
    - <packagedElement name="mobilephoto" xmi:id="
      ab61f56283e4" xmi:type="uml:Package">
      - <packagedElement name="ui" xmi:id="U59e52
        e106dc93399" xmi:type="uml:Package">
        + <packagedElement name="controller" xmi:
          ad2d-bfef682e28fd" xmi:type="uml:Pack
        - <packagedElement name="datamodel" xmi
          bb7e-35b879e65d45" xmi:type="uml:Pac
          - <packagedElement name="AlbumData
            93bd-9306055f6e4c" xmi:type="uml:
            + <ownedAttribute name="imageAcc
              4599-8cd9-df8978595e5f" type="
              0c3a60712ce3" xmi:type="uml:Pr
            + <ownedAttribute name="imageInfo
              461c-bcff-bc1edc80c2f5" type="U
              c226af8d4585" xmi:type="uml:Pr
```

Query: Interaction=MusicAlbumData()

Variation Block:

```
Lifeline=mediaAccessor:MusicMediaAccessor
Lifeline=MusicAlbumData
Lifeline=MediaData
Lifeline=mediaAccessor:MediaAccessor
Lifeline=SelectMediaController
Lifeline=MusicPlayController
Lifeline=MediaListController
Lifeline=MediaController
Message=MediaAccessor()
Message=MusicMediaAccessor()
Message=MediaListController()
Interaction=createNewAlbum(in albumName:String):void
Interaction=MusicAlbumData()
Interaction=addNewMediaToAlbum(in label:String, in path:String, in album:String
Interaction=SelectMediaController(in midlet:MainUIMidlet, in imageAlbumData:Al
Interaction=MusicPlayController(in midlet:MainUIMidlet, in albumData:AlbumDat
Interaction=MediaListController(in midlet:MainUIMidlet, in albumData:AlbumData
Interaction=MediaController(in midlet:MainUIMidlet, in albumData:AlbumData, in
```

Figure 7: SPL Design tool

To compare the performance of our feature, we used the metrics originally proposed by Bagheri et al. [29] and from which we took the following list:

- Number of features (NF): Counts the number of features in a feature model.
- Number of top features (NTop): Counts the number of features that are first direct descendants of the feature model root.
- Number of leaf features (NLeaf): Counts the number of features with no children or further specializations.
- Cyclomatic complexity (CC): Counts the number of distinct cycles that can be found in the feature model. Since feature models are in the form of trees, no cycles can exist in a feature model; however, integrity constraints between features can cause cycles. This metric counts the number of “exclude” and “require”.
- Ratio of variability (RoV): Counts the ratio of the average branching factor of the parent features in the feature model. In other words, the average number of children of the nodes in the feature model tree.
- Flexibility of configuration (FoC): Counts the ratio of the number of optional features over all of the available features in the feature model.
- Coefficient of connectivity density (CoC): Counts the ratio of the number of edges over the number of features in a feature model.

Figure 8 shows a comparison of quality metrics values obtained for the feature model of TankWar game,MobileMedia system, BerkeleyDB system, Soduko game and Acrade Game Maker.

It is clear that the values obtained by our approach are close to those obtained for the feature model resulting from the work of experts. The number of features in our feature models is nearby those belonging to feature models built by experts. The difference between the number of feature do not exceed 3 features. For example, in the feature model "TankWar", we find 35 feature whereas the expert found 37.

	FM	Metrics							
		NF	NTop	NLeaf	CC	RoV	CoC	FoC	DT
TankWar	*FM1*	37	9	25	7	4	1,27	0.216	4
	Our FM	35	8	21	6	3,6	1,1	0.3	5
MobileMedia	*FM2*	26	3	16	1	2,5	1,34	0,364	4
	Our FM	28	4	17	2	2,6	1	0,5	4
BerkeleyDB	*FM3*	47	7	35	2	4	3,67	1,42	4
	Our FM	46	7	9	1	2,2	2,52	1,41	5
Sodoku	*FM4*	30	9	18	2	3,6	1,21	0,368	4
	Our FM	28	10	15	3	4	1,1	0,40	4
AcradeGameMaker	*FM5*	13	4	9	1	3,1	0,975	0,25	3
	Our FM	11	3	14	1	3	1.7	0.65	3

Figure 8: A comparative study by measurement

In conclusion, our preliminary empirical study shows that the feature models generated are of high quality because they do not go beyond the values of the used metrics applied on existing feature models.

6 Conclusion

This paper presented a new bottom-up method for *automatically* extracting both the feature model and design of an SPL from product variants. Our method has the advantage of using semantic information to account for the differences in the structures and element names of the product variants. It first harmonizes the names of the source codes' elements. Then, it uses the FCA technique to distinguish among the mandatory and optional elements. To extract features with the appropriate cohesion, we first conducted a comparative study between three popular textual similarity measures (LSI, PCA and TF-IDF) and then adopted LSI for our feature identification process. As for the SPL design, our method produces SPL designs enriched with information extracted from the feature model and specified with our UML profile named SPL_UML. The utility of the proposed method is illustrated through the extraction of the feature model and design of an SPL for mobile phones. As presented in this paper, our method was quantitatively evaluated on five existing FM in different domains and it was compared to existing FM developed by experts. The precision and recall produced in our experimental evaluation showed the advantages of our method.

In our future works, we are examining how to add more intelligence in the feature model extraction by considering product variants where the variability is in the body of the operations. We aim also to conduct an evaluation on a larger set of products.

References

[1] P. Clements and L. Northrop, "Software product lines: Practices and patterns.," *SEI Series in Software Engineering*, 2001.

[2] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-oriented domain analysis (foda) feasibility study,," *Technical report CMU/SEI-90-TR-21, Software Engineering Institute,Carnegie Mellon University,*, 1990.

[3] T. Ziadi, L. Frias, M. Silva, and M. Ziane, "Feature identification from the source code of product variants," pp. 417–422, 2012.

[4] S. She, R. Lotufo, T. Berger, A. Wsowski, and K. Czarnecki, "Reverse engineering feature models," pp. 461–470, 2011.

[5] R. Al-Msie'Deen, A. Seriai, M. Huchard, C. Urtado, S. Vauttier, and H. Salman, "An approach to recover feature models from object-oriented source code," in *Day Product Line 2012*, 2012.

[6] T. Ziadi, "Manipulation de lignes de produits en uml," *These de doctorat, Universite de Rennes 1*, 2004.

[7] H. Salman, A. Seriai, C. Dony, and R. Al-Msie'Deen, "Genetic algorithms as recovering traceability links method between feature models and source code of product variants," in *Day Product Line 2012*, 2012.

[8] M. Acher, B. Baudry, P. Heymans, A. Cleve, and J.-L. Hainaut, "Support for reverse engineering and maintaining feature models," in *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS '13, (New York, NY, USA), pp. 1–8, 2013.

[9] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," pp. 226–231, AAAI Press, 1996.

[10] B. Ganter and R. Wille, "Formal concept analysis: Mathematical foundations," *Springer-Verlag*, 1996.

[11] D. Binkley and D. Lawrie, "Information retrieval applications in software maintenance and evolution," *In Encyclopedia of Software Engineering*, pp. 454–43, 2011.

[12] K. Czarnecki and U. Eisenecker, *Generative programming - methods, tools and applications.* Addison-Wesley, 2000.

[13] P. Riebisch, "Using feature modeling for program comprehension and software architecture recovery," (Huntsville Alabama, USA), 2003.

[14] N. Nan and E. Steve, "Concept analysis for product line requirements," in *Proceedings of the 8th ACM international conference on Aspect-oriented software development*, AOSD '09, (New York, NY, USA), pp. Pages 137–148, 2009.

[15] Z. Tewfik, H. Christopher, P. Mike, Z. Mikal, and L. Yves, "Towards a language-independent approach for reverse-engineering of software product lines," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pp. 1064–1071, 2014.

[16] M. Couto, M. Valente, and F. Figueiredo, "Extracting software product lines: A case study using conditional compilation," pp. 191–200, 2011.

[17] R. Al-Msie'deen, A. Seriai, M. Huchard, C. Urtado, S. Vauttier, and H. Salman, "Feature location in a collection of software product variants using formal concept analysis," in *Safe and Secure Software Reuse - 13th International Conference on Software Reuse, ICSR 2013, Pisa, Italy, June 18-20. Proceedings*, pp. 302–307, 2013.

[18] R. Al-Msie'deen, A. Seriai, M. Huchard, C. Urtado, and S. Vauttier, "Documenting the mined feature implementations from the object-oriented source code of a collection of software product variants," in *The 26th International Conference on Software Engineering and Knowledge Engineering, Hyatt Regency, Vancouver, BC, Canada, July 1-3, 2013.*, pp. 138–143, 2014.

[19] H. Salman, A. Seriai, and C. Dony, "Feature-level change impact analysis using formal concept analysis.," *International Journal of Software Engineering and Knowledge Engineering*, vol. 25, no. 1, pp. 69–92, 2015.

[20] H. Ben-Abdallah, N. Bouassida, F. Gargouri, and A. B. Hamadou, "A uml based framework design method," *Journal of Object Technology*, pp. 97–120, 2004.

[21] J. Maazoun, N. Bouassida, and H. Ben-Abdallah, "Feature model extraction from product source codes based on the semantic aspect," *ICSOFT13*, pp. 154–161, 2013.

[22] J. Maâzoun, N. Bouassida, and H. Ben-Abdallah, "Feature model recovery from product variants based on a cloning technique," in *The 26th International Conference on Software Engineering and Knowledge Engineering, Hyatt Regency, Vancouver, BC, Canada, July 1-3, 2013.*, pp. 431–436, 2014.

[23] S. Dumais, “Latent semantic indexing (lsi) and trec-2,” in *The Second Text REtrieval Conference (TREC-2*, pp. 105–115, 1994.

[24] K. Jones, “A statistical interpretation of term specificity and its application in retrieval,” *Journal of Documentation*, vol. 28, pp. 11–21, 1972.

[25] I. Jolliffe, *Principal Component Analysis*. Springer Verlag, 1986.

[26] L. Smith, *A tutorial on Principal Component Analysis*. 2002.

[27] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten, “The weka data mining software: An update,” in *SIGKDD Exploration*, vol. 11, 2009.

[28] J. Maazoun, N. Bouassida, and H. Ben-Abdallah, “A bottom up spl design method,” *MODELSWARD’14*, (Janvier 2014).

[29] E. Bagheri and D. Gasevic, “Assessing the maintainability of software product line feature models using structural metrics,” in *Software Quality Journal*, vol. 19, pp. 579–612, 2011.

Experiences on Writing Ada Bindings for a C Library

Victor Porton
Ashkelon, Israel; email: porton@narod.ru

Abstract

I share my experience of writing thick (object oriented) Ada binding of a C library. This article provides some Ada tips and tricks (especially for Ada bindings writers).

Keywords: Ada bindings, thick bindings, C.

1 Introduction

We have a C library, written in object-oriented style (C structure pointers serve as objects, and C functions taking such structure pointers serve as methods). However fortunately for us (making our task easier) there is no inheritance in that C library.

The particular libraries we will deal with are *Redland RDF Libraries*, a set of libraries which parses RDF (Resource Description Framework) files or other RDF resources and manages them, allows to do RDF queries, etc. Don't worry if you don't know what RDF is, it is not really relevant for this article. See more info about this C library in [1] and [2] and on RDF itself in [3].

The paper describes *thick* Ada bindings for this library. “Thick” means that the API which I create is a full-fledged Ada interface. For example, it uses Ada controlled tagged types to represent objects. It also uses derived types and some other Ada features which are not available in C. By Ada here I will mean Ada 2012, the latest currently available Ada standard.

This is a work in progress. The source code of my library is available at https://github.com/vporton/redland-bindings. Please write your comments.

Note that the main purpose I created Ada bindings for Redland is to use them in this project: https://en.wikiversity.org/wiki/Automatic_transformation_of_XML_namespaces.

2 Little things

One thing I learned during this project, is that Ada types should have different names, they shouldn’t have the same name even if they are in different packages. This both allows to shorten the code with use directives and to increase reliability of not passing a wrong type if a use directive is indeed used.

An alternative solution would be to use renames of packages. But then not only types but also subprograms would be to be referred with a prefix. However my actual solution allows to use subprogram overloading to write subprograms without prefixes, what personally I consider more convenient. Also (and probably more importantly), even if I were recommended to use package renames, not “use” directives, some users of my code would probably use “use” directives and get into difficulties with coinciding types with the same name from different packages.

The reason why I prefer this solution (to have different types names in different packages), is that this allows the user the freedom to choose at his mood either “use” directives or package renames. He is not bound to use package renames.

Initially I tried to use GCC with `-fdump-ada-spec` flag to autogenerate Ada specs from C headers. But shortly I realized that it will work better if I write Convention=>C subprograms manually (mainly because I sometimes want char_array and sometimes chars_ptr for a char* argument of a C function, as char_array is useful and convenient for passing the value of To_C function to a subprogram).

3 Packages structure

I put all my API into package hierarchy RDF.*.

The package RDF itself is empty:

```
package RDF is
  pragma Pure(RDF);
end RDF;
```

I also have a RDF.Auxiliary package and its subpackages for “auxiliary” things (things used by or with my bindings, but not being bindings for a particular C library function).

I will discuss some particular RDF.Auxiliary.* packages below.

4 My tagged types

As I've said above, C objects are pointers to structures. All C pointers to structures have the same format and alignment [4]. This allows to represent any pointers to C structures as pointers to Dummy_Record as defined in RDF.Auxiliary package:

```
type Dummy_Record is null record
  with Convention=>C;
```

A new Ada type (call it T_Without_Finalize for the reasons explained below) corresponding to a dynamically allocated C record is created by instantiating generic packages RDF.Auxiliary.Handled_Record or RDF.Auxiliary. Limited_Handled_Record with a Convention=>C record type (can be Dummy_Record if record layout is considered internal by the C library documentation) and a Convention=>C access to this record and deriving our type from the tagged type Base_Object in the instantiated package.

Representing C structure pointers as tagged types is not memory efficient, but here we trade efficiency for programming ease.

It would be compelling to make Limited_Handled_Record a descendant type of Handled_Record, but it is impossible in Ada because Ada.Finalization.Limited_Controlled is not a descendant of Ada.Finalization.Controlled. As such I wrote two similar packages RDF.Auxiliary. Limited_Handled_Record and RDF.Auxiliary.Handled_ Record which duplicate mainly the same code. This is not perfect, but neither it is very bad, as the quantity of the code of these two packages (including their bodies) is not large.

5 About finalization and related stuff

The main challenge writing object-oriented bindings for a C library is *finalization*. In the C library in consideration (as well as in many other C libraries) every object is represented as a pointer to a dynamically allocated C structure.

The corresponding Ada object can be a (tagged) record holding the pointer (aka *handle*).

Ada objects representing C objects should be descendants of Ada.Finalization.Limited_Controlled or Ada.Finalization. Controlled to be properly finalized when appropriate.

But oftentimes a C function returns a so called “shared handle” that is a pointer to a C struct which we should not free because it is a part of a larger C object and shall be freed (by the C library) only when that larger C object goes away.

As such I first define a tagged type T_Without_Finalize type. For this type I define such procedures as Do_Finalize and Do_Adjust which do what Finalize and Adjust should do but leave Finalize and Adjust empty, so that a shared handle is neither finalized nor copied. I define type T with Finalize and Adjust as a derived type. T could be defined as ancestor of both T_Without_Finalize and a type which defines Finalize and Adjust. But as Ada misses inheritance from multiple tagged private types, I do it with generics instead (below is a partial listing):

```
generic
  type Base is new Base_Object with private;
package Common_Handlers is
  type User_Type is new Base with null record;
  overriding procedure Finalize(Object: in out User_Type)
                          renames Do_Finalize;
  overriding procedure Adjust(Object: in out User_Type)
                        renames Do_Adjust;
  type Base_With_Finalization is new User_Type
    with null record;
end;
```

The Base generic parameter is intended to be that T_Without_Finalize type.

You see that Do_Finalize and Do_Adjust become actual handlers of finalization and adjustment.

Note that I recommend overriding the subprograms Finalize_Handle and Adjust_Handle (see the source) rather than Do_Finalize and Do_Adjust themselves.

Note that values of T_Without_Finalize type may become invalid (containing dangling access values). There seems that there is no easy enough way to deal with this problem (because of the way the C library works). The objects are sometimes destroyed by the C library and we may not know when it is destroyed. Or we may know but be not able to appropriately “explain” it to the Ada compiler. Just be careful when using this library not to use objects which are already destroyed.

6 User defined types

Next thing to note that I first define User_Type. This type is intended to serve among other as a base for user-defined types which may contain not only the C handle but also other fields. The type Base_With_Finalization on the other hand is meant not to be a base for types with additional fields but contain only the handle (and null record extensions).

The reason why I make distinction between User_Type and Base_With_Finalization is the following:

We define some functions like

```
function From_Filename
  (World: Raptor_World_Type_Without_Finalize'Class;
   Filename: String)
    return IOStream_Type;
```

IOStream_Type is derived from Base_With_Finalization not from User_Type directly. If we derived our User_Type from IOStream_Type then non-null record extensions would cause (by Ada rules) the necessity to redefine From_Filename function also for the derived type, which is a nonsense.

We actually use User_Type (in the private part of a package) like this (for an I/O stream reading from a string):

```
type Stream_From_String(Length: size_t) is
  new IOStream_Type_User with
  record
    Str: char_array(1..Length);
  end record;
```

7 Controlling vs class-wide arguments

Controlling and class-wide arguments differ mainly in their relationship with inheritance. But as there is no inheritance in the C library which we bind, we have certain freedom to choose either.

One disadvantage of class-wide types is that such things as that is makes necessary Get_Handle(null) to be type-qualified and thus the subprogram specifications longer. That is we need to use something like URI_Type_Without_Finalize'(From_Handle(null)) or

URI_Type'(From_Handle(null)) instead of just From_Handle(null) to make it unambiguous (as otherwise the Ada compiler rejects it with a compilation error).

One advantage of class-wide types is that I can use (as in query_results.ads) ST'Class where ST is a subtype with a predicate to restrict to a subtypes matching a predicate.

Example:

```
subtype URI_Term_Type_Without_Finalize is
 Term_Type_Without_Finalize
  with Dynamic_Predicate =>
    Is_URI(URI_Term_Type_Without_Finalize);
```

It is possible that in a future version of the library I will consistently replace controlling arguments with class-wide arguments. This would make it more symmetric, as all tagged arguments would be class-wide and none special controlling one.

8 Dealing with callbacks

To deal with C callbacks (particularly accepting a void* argument for additional data) in object-oriented way, we need a way to convert between C void pointers and accesses to Ada tagged (even class-wide) objects. (We pass Ada tagged objects as C "user data" pointers.)

When we create a callback we need to pass an Ada object as a C pointer and a Convention=>C subprogram defined by us as the callback. The callback receives the pointer previously passed by us and in the callback code we should (if we want to stay object oriented) convert this pointer into an Ada object access.

What we need is some bijective ("back and forth") mapping between Ada access values and C pointers.

At first, I was tempted to use Ada.Unchecked_Conversion. But (despite GNAT 7.2.0 gives no warning on this) it is not in any way guaranteed to work, because the format of Ada access type and of C pointer are not necessarily the same.

Now I do conversion this way:

I convert chars_ptr to a Convention=>C access to char then this to System.Address using System.Address_ To_Access_Conversions and then the address (also by Address_To_ Access_Conversions) to the required access to a class-wide type.

The backward conversion is analogous.

The above should work if we understand the words "back and forth" RM13.7.2(5/2) "The To_Pointer and To_Address subprograms convert back and forth between values of types Object_Pointer and Address." as that the conversion must be bijective (I filed a clarification request [1] about the meaning of the words "back and forth" to the Ada Commentaries).

[1] AC95-00298/00, discussion and answer available at http://www.ada-auth.org/cgi-bin/cvsweb.cgi/acs/ac-00298.txt?rev=1.1

All this is implemented in RDF.Auxiliary.Convert_Void of my library, but in my opinion this package should be added to Ada standard packages.

How to do this in practice? The best way to explain is an example (for a user-defined I/O Stream which calls our function Do_Write_Bytes when "write" message is sent to it):

```
package My_Conv is
  new RDF.Auxiliary.Convert_Void
    (Handled_IOStream_Type_User'Class);
function raptor_iostream_write_bytes_impl
  (context: chars_ptr; ptr: chars_ptr; size, nmemb: size_t)
   return int
  with Convention=>C;
function raptor_iostream_write_bytes_impl
  (context: chars_ptr; ptr: chars_ptr; size, nmemb: size_t)
   return int is
begin
  declare
      Result: constant int := Do_Write_Bytes
(My_Conv.To_Access (context).all, ptr, size, nmemb);
  begin
      return Result;
  end;
exception
  when others =>
      return -1;
end;
```

9 Storage pools for memory allocation

I tried to define storage pools for C allocation/deallocation functions such as raptor_alloc_memory() and raptor_free_memory(), but it appeared to be impossible by the following reason:

System.Storage_Pools receives Alignment argument which is an integer multiple of the alignment of the allocated type. This alignment may be greater than the alignment raptor_alloc_memory() warrants (Dummy_Record' Alignment) and so lead to undefined behavior.

I have sent a proposal to the standardization committee [2] to make the programmer able to restrict the maximum alignment.

Because using allocators appeared to be impossible, I did it instead this way (for Locator_Handle which is a pointer to Locator_Type record):

```
package Locator_Conv is
  new RDF.Auxiliary.Convert_Void(Locator_Type_Record);

function Copy_Locator (Handle: Locator_Handle)
  return Locator_Handle
is
  Size: constant size_t :=
size_t((Locator_Type'Max_Size_In_Storage_Elements *
Storage_Unit + (char'Size-1)) / char'Size);
  Result2: constant chars_ptr :=
    RDF.Raptor.Memory.raptor_alloc_memory(Size);
```

[2] AC95-00299/00, discussion and answer available at www.ada-auth.org/cgi-bin/cvsweb.cgi/acs/ac-00299.txt?rev=1.1

```
  Result: constant Locator_Handle :=
    Locator_Handle(Locator_Conv.To_Access(Result2));
begin
  Result.all := Handle.all;
  Result.URI := raptor_uri_copy(Handle.URI);
  Result.File :=
RDF.Raptor.Memory.Copy_C_String(Handle.File);
  return Result;
end;
```

Note that (Locator_Type'Max_Size_In_Storage_Elements * Storage_Unit + (char'Size-1)) / char'Size is the ceiling of floating point division of Locator_Type'Max_Size_In_ Storage_Elements * Storage_Unit by char'Size (but without using floating point). Using ceiling warrants that the allocated space is at least as big as required space.

Here I allocate with raptor_alloc_memory() function the amount of memory which is max size needed (apparently not to overwrite nearby memory) for a record pointed by Locator_Type (ARM specifies this max size only for memory returned by an allocator, but I am pretty sure that in any reasonable implementation of Ada the same amount of memory will work well if it is allocated by raptor_alloc_memory() function instead and the nearby memory thus won't be overwritten).

10 More little things

Ada standard misses a function converting a C string (with possible NULs) described by a chars_ptr and and its length in characters into an Ada String. There is the function To_Ada which accepts a char_array argument. But in real life we get a char pointer (for example of chars_ptr type) and its length, not an array from a C library, so we cannot use To_Ada in some circumstances.

I define function Value_With_Possible_NULs which does this in terms of Interfaces.C.Pointers. Note that the pointer defined in suitably instantiated Interfaces.C.Pointers is correctly converted from/to chars_ptr with Ada.Unchecked_Conversion.

I also gave the proposal [3] to add such a function to Ada standard, in order not to re-create it every time one writes a bindings of a C library. See the discussion at this proposal for more details.

The Ada standard To_C with Trim_Nul=>False is broken (accordingly my personal opinion): RM B.3(51) "If Append_Nul is False and Item'Length is 0, then To_C propagates Constraint_Error." Said in another way the Standard means: "This does not work with empty strings." So I wrote a wrapper My_To_C_Without_Nul around it. It's a fact that there are met empty (non-null-terminated) strings in real life and we need to deal with them. An example of such a circumstance is the result of reading an external file: It may be empty and it may contain NUL chars.

I would write a lot more advice how to write Ada bindings for a C library, but you can just follow my source, which can serve as an example.

I "encode" values of C strings (which can be NULL) as an Ada indefinite holder holding a String. If the string is NULL, the holder is empty. However often it is enough to transform an empty Ada string into NULL C string (this can work only if we don't differentiate between empty and null strings).

References

[1] The Design and Implementation of the Redland RDF Application Framework. David Beckett, 2001, http://www10.org/cdrom/papers/490/

[2] Bootstrapping RDF applications with Redland. David Beckett, https://www.dajobe.org/papers/xtech2005/

[3] Resource Description Framework (RDF). RDF Working Group, https://www.w3.org/RDF/

[4] ISO/IEC 9899:2011, section 6.2.5, paragraph 28.

[3] AC95-00291/00, discussion and answer available at www.ada-auth.org/cgi-bin/cvsweb.cgi/acs/ac-00291.txt?rev=1.2

Applied Formal Logic: Searching in Strings*

Yannick Moy
AdaCore, France

A friend pointed me to recent posts by Tommy M. McGuire (TMM), in which he describes how Frama-C can be used to functionally prove a brute force version of string search [1], and to find a previously unknown bug in a faster version of string search called quick search [2]. Frama-C and SPARK share similar history, techniques and goals. So it was tempting to redo the same proofs on equivalent code in SPARK, and completing them with a functional proof of the fixed version of quick search. This is what I'll present in this post.

Contrary to strings in C which start at index 0, standard strings in SPARK range over positive numbers, and usually start at index 1. I could have made my own strings to start at index 0, but there is no reason to stick to C convention when writing the algorithm in SPARK. At the same time, it's convenient to force the string to start at index 1 with an explicit predicate, which I do like that:

```
subtype Text is String with Predicate => Text'First = 1;
```

Following the order of exposure of Tommy M. McGuire's posts, here is the implementation for the brute force algorithm in SPARK:

```
function Brute_Force (Needle, Haystack : in Text)
     return Natural is
   Diff : Boolean;
begin
   for I in 1 .. Haystack'Length - Needle'Length + 1
loop
      Diff := False;

      for J in Needle'Range loop
         Diff := Needle(J) /= Haystack(J + (I - 1));
         exit when Diff;
      end loop;

      if not Diff then
         return I;
      end if;
   end loop;
   return 0;
end Brute_Force;
```

I am doing here without the parameters n and h which were used in the C version to denote the length of strings needle and haystack, since these are readily available as attributes Haystack'Length and Needle'Length in SPARK. Since I'm working on strings starting at index 1, there are a few adjustments compared to the C version. The use of a temporary variable Diff is needed to detect that the inner loop was exited due to a difference between Needle and the portion of Haystack starting at J, as the for-loop in SPARK does not increment its index in the last iteration of the loop, contrary to its C version.

On this initial version, GNATprove issues one message about a possible integer overflow when computing "Haystack'Length - Needle'Length + 1". It automatically proves all other run-time checks (2 initialization checks, 1 array index check, 2 integer range checks, 2 integer overflow checks). GNATprove also provides a counterexample to understand the possible failure, which can be displayed in our IDE GPS by clicking on the magnify icon on the left of the message/line (Figure 1).

You have to scroll right in the IDE to see all the values, so here are the relevant ones: Haystack'First = 1 and Haystack'Last = 2147483647 and Needle'First = 1 and Needle'Last = 0. In that case, Haystack'Length is 2147483647 and Needle'Length is 0, which means that "Haystack'Length - Needle'Length + 1" is one past the largest signed 32-bits integer. Hence the overflow. One way to avoid this issue is to require that Needle is not the empty string, so its length is at least 1:

```
function Brute_Force (Needle, Haystack : in Text)
      return Natural with
  Pre => Needle'Length >= 1;
```

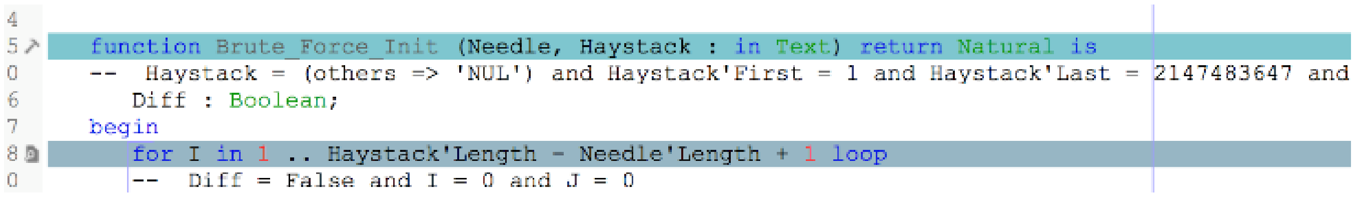
This precondition is sufficient for GNATprove to prove all checks in Brute_Force, but I've made it stronger like done by TMM in his post, as it does not make sense to look for a needle that is longer than the haystack:

```
function Brute_Force (Needle, Haystack : in Text)
      return Natural with
  Pre => Needle'Length in 1 .. Haystack'Length;
```

* Paper derived from blog post at http://www.spark-2014.org/entries/detail/applied-formal-logic-searching-in-strings

[1] https://maniagnosis.crsr.net/2017/06/AFL-brute-force-search.html

[2] https://maniagnosis.crsr.net/2017/06/AFL-bug-in-quicksearch.html



```
function Brute_Force_Init (Needle, Haystack : in Text) return Natural is
--  Haystack = (others => 'NUL') and Haystack'First = 1 and Haystack'Last = 2147483647 and
   Diff : Boolean;
begin
   for I in 1 .. Haystack'Length - Needle'Length + 1 loop
   --  Diff = False and I = 0 and J = 0
```

Figure 1. Magnify icon in GPS (left of the message/line)

Note that, compared to what is needed with Frama-C, we don't need here to provide loop assigns or loop invariants. GNATprove automatically computes the variables that are modified in a loop, as well as the range of for-loop indexes. Still following the order of exposure of TMM's posts, let's turn to the functional contract for searching a string. I'm directly translating here the functions partial_match_at and match_at given by TMM from C to SPARK, as well as the contract of brute_force. Functions Partial_Match_At and Match_At are ghost functions in SPARK (with aspect Ghost), which means that they can be used only in assertions/contracts and ghost code [3]. A difference with Frama-C is that ghost code is executable like regular code in SPARK, so one must show absence of run-time errors in ghost code as well, hence the precondition on Partial_Match_At below:

```
--  There is a partial match of the needle at location
--  loc in the haystack, of length len.
function Partial_Match_At   (Needle, Haystack : Text;
                             Loc : Positive; Len : Natural)
         return Boolean
is
  (for all I in 1 .. Len => Needle(I) =
                                Haystack(Loc + (I - 1)))
with Ghost,
     Pre => Len <= Needle'Length
      and then Loc - 1 <= Haystack'Length - Len;

--  There is a complete match of the needle at location
--  loc in the haystack.
function Match_At (Needle, Haystack : Text;
                   Loc : Positive) return Boolean is
  (Loc - 1 <= Haystack'Length - Needle'Length
   and then Partial_Match_At (Needle, Haystack,
                               Loc, Needle'Length))
with Ghost;
```

The contract on Brute_Force is similar to the one in Frama-C, with a shift by one for the origin of strings, Brute_Force'Result instead of \result to denote the result of the function, and an if-expression instead of behaviors (SPARK has a similar notion of contract cases [4], but they must always have disjoint guards in SPARK, so are not applicable here):

[3] http://docs.adacore.com/spark2014-docs/html/ug/en/source/specification_features.html#ghost-code

[4] http://www.spark-2014.org/entries/detail/spark-2014-rationale-contract-cases

```
function Brute_Force (Needle, Haystack : in Text)
         return Natural with
  Pre  => Needle'Length in 1 .. Haystack'Length,
  Post => Brute_Force'Result in 0 ..
                  Haystack'Length - Needle'Length + 1
   and then
   (if Brute_Force'Result > 0 then
     Match_At (Needle, Haystack, Brute_Force'Result)
    else
     (for all K in Haystack'Range =>
       not Match_At (Needle, Haystack, K)));
```

Before we even try to prove that this contract is satisfied by the implementation of Brute_Force, it is a good idea to test it on a few inputs, to get rid of silly mistakes. Here is a test driver to do precisely that:

```
with String_Search; use String_Search;
procedure Test_Search is
   All_Men : constant Text :=
     "We hold these truths to be self-evident, that all men
       are created equal,"
     & " that they are endowed by their Creator with
         certain unalienable "
     & "Rights, that among these are Life, Liberty and the
        Pursuit of "
     & "Happiness. That to secure these rights,
        Governments are instituted "
     & "among Men, deriving their just powers from the
        consent of the governed";
begin
   pragma Assert (
               Brute_Force (All_Men, "just powers") > 0);
   pragma Assert (
               Brute_Force (All_Men, "austin powers") = 0);
end Test_Search;
```

Just compile the code with assertions on (switch -gnata), run it, and... it fails the precondition of Brute_Force:

```
raised SYSTEM.ASSERTIONS.ASSERT_FAILURE :
failed precondition from string_search.ads:24
```

What happened here is that I put arguments in the wrong order in the call to Brute_Force. I'm not making this up, this really happened to me (I am that bad!). Anyway, that illustrates that testing is a good idea, even if here it detected a bug in the test itself. The fix in SPARK is to use named parameters to avoid such issues. They don't have to appear in the same order as in the function signature, but it's a good idea nonetheless:

```
pragma Assert (Brute_Force (Needle => "just powers",
                            Haystack => All_Men) > 0);
```

Once fixed, the test passes without errors. Like in the case of Frama-C, we need to add loop invariants for GNATprove to prove that Brute_Force satisfies its contract. Loop invariants in SPARK are different from the classical loop invariants used in Frama-C: you can put them anywhere in the loop, and they don't have to hold when reaching/exiting the loop but only when execution reaches the program point of the loop invariant. I prefer in general to put loop invariants at the end of loops, because it's more natural to express what has been achieved so far:

```
function Brute_Force (Needle, Haystack : in Text)
       return Natural is
   Diff : Boolean;
begin
   for I in 1 .. Haystack'Length - Needle'Length + 1
           loop
     Diff := False;

     for J in Needle'Range loop
       Diff := Needle(J) /= Haystack(J + (I - 1));
       exit when Diff;
       pragma Loop_Invariant (
              Partial_Match_At (Needle, Haystack, I, J));
       pragma Loop_Invariant (
              Diff = (Needle(J) /= Haystack(J + (I - 1))));
     end loop;

     if not Diff then
       return I;
     end if;

     pragma Loop_Invariant
       (for all K in 1 .. I => not Match_At (
                                  Needle, Haystack, K));
   end loop;

   return 0;
end Brute_Force;
```

A subtlety above is that, since we're replacing the implicit loop invariant in the inner loop (located at the start of the loop) by an explicit loop invariant at the end of the inner loop, we need to repeat in that loop invariant the information about the current value of Diff, otherwise this information is not available on the path starting from the loop invariant and exiting the loop in the last iteration. Otherwise this is similar to what was done in Frama-C. With these loop invariants, GNATprove proves all checks in Brute_Force, including its postcondition.

I kept above the implementation structure originating from the C version of Brute_force, but in SPARK we can simplify it by replacing the inner loop with a direct comparison of Needle with a slice of Haystack:

```
function Brute_Force (Needle, Haystack : in Text)
       return Natural is
begin
  for I in 1 .. Haystack'Length - Needle'Length + 1 loop
     if Needle = Haystack(I .. I + (Needle'Last - 1)) then
       return I;
     end if;

     pragma Loop_Invariant
       (for all K in 1 .. I => not Match_At (Needle,
                                  Haystack, K));
   end loop;

   return 0;
end Brute_Force;
```

This version is also completely proved by GNATprove.

Now turning to the more involved algorithm for string search called quick search presented in this other post by TMM [5]. Translating the implementation, contracts and loop invariants in SPARK is quite easy. As for the brute force version, more precise types in SPARK allow to get rid of a number of annotations:

```
type Shift_Table is array (Character) of Positive;

procedure Make_Bad_Shift (Needle : Text;
              Bad_Shift : out Shift_Table) with
  Pre  => Needle'Length < Integer'Last,
  Post => (for all C in Character => Bad_Shift(C)
                 in 1 .. Needle'Length + 1);

function QS (Needle, Haystack : in Text)
             return Natural with
  Pre => Needle'Length < Integer'Last
   and then Haystack'Length < Integer'Last - 1
   and then Needle'Length in 1 .. Haystack'Length;
```

I am also getting rid of a loop in Make_Bad_Shift and a loop in QS compared to their C version, as we can directly assign and compare strings in SPARK:

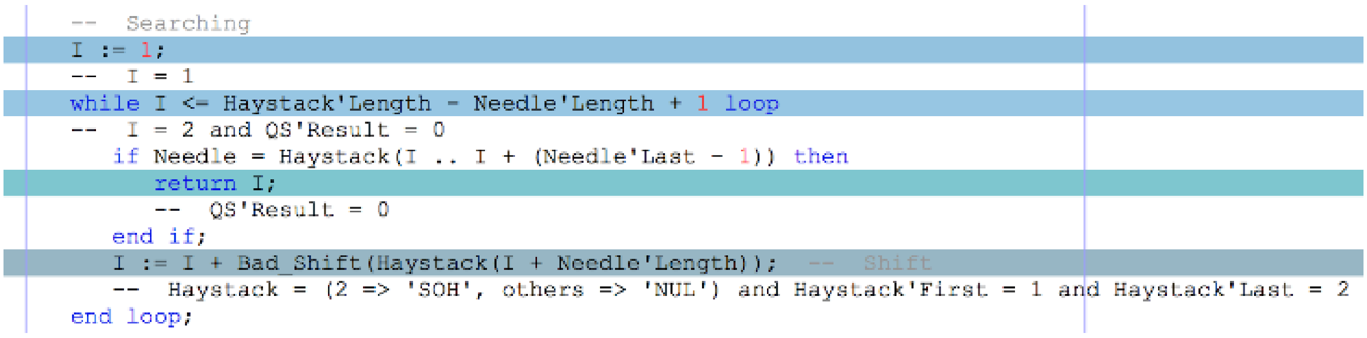
```
procedure Make_Bad_Shift (Needle : Text;
                  Bad_Shift : out Shift_Table) is
begin
   Bad_Shift := (others => Needle'Length + 1);

   for J in Needle'Range loop
     Bad_Shift(Needle(J)) := Needle'Length - J + 1;
     pragma Loop_Invariant (
        for all C in Character => Bad_Shift(C) in
                      1 .. Needle'Length + 1);
   end loop;
end Make_Bad_Shift;

function QS (Needle, Haystack : in Text)
     return Natural is
   Bad_Shift : Shift_Table;
   I : Positive;

begin
   -- Preprocessing
   Make_Bad_Shift (Needle, Bad_Shift);
```

[5] https://maniagnosis.crsr.net/2017/06/AFL-bug-in-quicksearch.html



```
--  Searching
I := 1;
--  I = 1
while I <= Haystack'Length - Needle'Length + 1 loop
--  I = 2 and QS'Result = 0
   if Needle = Haystack(I .. I + (Needle'Last - 1)) then
      return I;
      --  QS'Result = 0
   end if;
   I := I + Bad_Shift(Haystack(I + Needle'Length));  --  Shift
   --  Haystack = (2 => 'SOH', others => 'NUL') and Haystack'First = 1 and Haystack'Last = 2
end loop;
```

Figure 2. Counter example

```
    --  Searching
    I := 1;
    while I <= Haystack'Length - Needle'Length + 1
                                               loop
      if Needle = Haystack(I .. I + (Needle'Last - 1)) then
        return I;
      end if;
      I := I + Bad_Shift(Haystack(I + Needle'Length));
      --  Shift
    end loop;
    return 0;
  end QS;
```

GNATprove proves all checks on the above code, including postconditions, except for the array index check when computing "Haystack(I + Needle'Length)". This is precisly the bug that was discovered by TMM, that he presents in his post. GNATprove further helps by providing a counterexample to understand the possible failure (Figure 2).

Indeed, when I=2 and Haystack'Last=2, "I + Needle'Length" is outside of the bounds of Haystack whenever Needle is not the empty string. We can fix that by exiting early from the loop before the assignment to I in the loop:

```
      exit when I = Haystack'Length - Needle'Length + 1;
```

With this fix, GNATprove proves all checks on the code of quick search.

Now turning to proving the functional behavior of quick search. The postcondition of QS is the same as the one of Brute_Force, given that only the algorithm changes between the two:

```
  function QS (Needle, Haystack : in Text)
          return Natural with
    Pre => Needle'Length < Integer'Last
      and then Haystack'Length < Integer'Last - 1
      and then Needle'Length in 1 .. Haystack'Length,
    Post => QS'Result in 0 ..
                Haystack'Length - Needle'Length + 1
      and then
      (if QS'Result > 0 then
         Match_At (Needle, Haystack, QS'Result)
       else
        (for all K in Haystack'Range =>
           not Match_At (Needle, Haystack, K)));
```

In order to prove the contract of QS, we'll need to specify and prove the functional behavior of Make_Bad_Shift first. As explained by TMM in his post, Make_Bad_Shift is used to align the last instance of a given character in the needle with a matching character in the haystack. So for every such character C, either it does not occur in the needle in which case Bad_Shift(C) has the value "Needle'Length + 1", or it occurs (possibly multiple times) in the needle in which case it occurs last at index "Needle'Length - Bad_Shift(C) + 1". This is what is expressed in the following postcondition:

```
  procedure Make_Bad_Shift (Needle : Text;
                    Bad_Shift : out Shift_Table) with
    Pre  => Needle'Length < Integer'Last,
    Post => (for all C in Character =>
                 Bad_Shift(C) in 1 .. Needle'Length + 1)
     and then (for all C in Character =>
            (if Bad_Shift(C) = Needle'Length + 1 then
              (for all K in
                    Needle'Range => C /= Needle(K))
             else
              Needle(Needle'Length -
                        Bad_Shift(C) + 1) = C
              and (for all K in Needle'Length -
                        Bad_Shift(C) + 2 ..
                   Needle'Last => Needle(K) /= C)
         ));
```

In order to prove that the implementation of Make_Bad_Shift satisfies this postcondition, we simply have to repeat this postcondition as a loop invariant, accumulating that information as the loop index J progresses (see how occurrences of Needle'Last in the postcondition were replaced by occurrences of J in the loop invariant):

```
  procedure Make_Bad_Shift (Needle : Text;
                    Bad_Shift : out Shift_Table) is
  begin
    Bad_Shift := (others => Needle'Length + 1);

    for J in Needle'Range loop
      Bad_Shift(Needle(J)) := Needle'Length - J + 1;
      pragma Loop_Invariant (for all C in Character =>
                 Bad_Shift(C) in 1 .. Needle'Length + 1);
      pragma Loop_Invariant (for all C in Character =>
           (if Bad_Shift(C) = Needle'Length + 1 then
                (for all K in 1 .. J => C /= Needle(K))
              else
                Needle(Needle'Length -
                                  Bad_Shift(C) + 1) = C
            and (for all K in Needle'Length -
                Bad_Shift(C) + 2 .. J => Needle(K) /= C)
                ));
      end loop;
    end Make_Bad_Shift;
```

GNATprove proves all checks on the above code.

Now turning to QS, we need to establish a loop invariant very similar to the one used in Brute_Force, except here we want to establish the property that Needle does not match up to index "I + Bad_Shift(Haystack(I + Needle'Length)) - 1" instead of just I:

```
     pragma Loop_Invariant
       (for all K in 1 .. I + Bad_Shift(Haystack(I +
                              Needle'Length)) - 1 =>
            not Match_At (Needle, Haystack, K));
```

We also need to bound I in the loop invariant, as we're inserting the above loop invariant in the middle of the loop, hence we do not get "for free" that I satisfies the loop test:

```
      pragma Loop_Invariant (I <= Haystack'Length -
                                   Needle'Length);
```

With these additions, GNATprove proves all checks in QS, including its postcondition, but it does not prove its loop invariant:

```
string_search.adb:111:81: medium: loop invariant might
fail after first iteration, cannot prove not Match_At
(Needle, Haystack, K) (e.g. when Haystack = (0 =>
'NUL', 5 => 'NUL', others => 'SOH') and Haystack'First =
1 and Haystack'Last = 6 and I = 4 and K = 5 and Needle
= (0 => 'SOH', 3 => 'SOH', 4 => 'SOH', 6 => 'SOH',
others => 'NUL') and Needle'First = 1 and Needle'Last =
2)
string_search.adb:111:81: medium: loop invariant might
fail in first iteration, cannot prove not Match_At (Needle,
Haystack, K) (e.g. when Haystack = (0 => 'NUL', 2 =>
'NUL', others => 'SOH') and Haystack'First = 1 and
Haystack'Last = 3 and I = 1 and K = 2 and Needle = (0
=> 'SOH', 3 => 'SOH', others => 'NUL') and Needle'First
= 1 and Needle'Last = 2)
```

This is expected. There is a big reasoning gap to go from the postcondition of Make_Bad_Shift to the loop invariant in QS. We are going to use ghost code to close that gap and convince GNATprove that the loop invariant holds in every iteration. What we need to show is that, for every starting position that is skipped (for K in the range I + 1 to I + Bad_Shift(Haystack(I + Needle'Length)) - 1), the needle cannot align with the haystack at that position. In fact, we know exactly at which position these alignments would fail: at the position "I + Needle'Length" in Haystack. Looking at the postcondition of Make_Bad_Shift, this corresponds to position "I + Needle'Length - K + 1" in Needle. Let's write it down just before the loop invariant:

```
    for K in I + 1 .. I + Bad_Shift(Haystack(I +
                          Needle'Length)) - 1 loop
      pragma Assert (Haystack(I + Needle'Length) /=
                     Needle(I + Needle'Length - K + 1));
      pragma Assert (not Match_At (Needle,
                                   Haystack, K));
    end loop;
```

GNATprove proves the above assertions, using the first one to prove the second one, so we can now accumulate this information in a loop invariant for all values of positions that are skipped:

```
    for K in I + 1 .. I + Bad_Shift(Haystack(I +
                        Needle'Length)) - 1 loop
      pragma Assert (Haystack(I + Needle'Length) /=
                   Needle(I + Needle'Length - K + 1));
      pragma Loop_Invariant
        (for all L in 1 .. K => not Match_At (Needle,
                                      Haystack, L));
    end loop;
```

With this addition of ghost code, GNATprove proves all checks in QS, including its postcondition and loop invariants. In the final version of that code, I'm using a local ghost procedure Prove_QS instead of inlining the ghost code in the implementation of QS. That way, GNATprove still internally inlines the implementation of Prove_QS to prove QS, but the compiler will completely get rid of the body and call to Prove_QS in the final executable built without assertions:

```
  function QS (Needle, Haystack : in Text)
          return Natural is
    Bad_Shift : Shift_Table;
    I : Positive;

    procedure Prove_QS with Ghost is
      Shift : constant Positive := Bad_Shift(Haystack(I +
                                    Needle'Length));
    begin
      for K in I + 1 .. I + Shift - 1 loop
        pragma Assert (Haystack(I + Needle'Length) /=
                   Needle(I + Needle'Length - K + 1));
        pragma Loop_Invariant
          (for all L in 1 .. K => not Match_At (Needle,
                                      Haystack, L));
      end loop;
    end Prove_QS;

  begin
    -- Preprocessing
    Make_Bad_Shift (Needle, Bad_Shift);

    -- Searching
    I := 1;
    while I <= Haystack'Length - Needle'Length + 1
     loop
      if Needle = Haystack(I .. I + (Needle'Last - 1)) then
        return I;
      end if;
      exit when I = Haystack'Length -
                                 Needle'Length + 1;

      Prove_QS;

      pragma Loop_Variant (Increases => I);
      pragma Loop_Invariant (I <= Haystack'Length -
                                    Needle'Length);
      pragma Loop_Invariant
        (for all K in 1 .. I + Bad_Shift(Haystack(I +
                              Needle'Length)) - 1 =>
              not Match_At (Needle, Haystack, K));

      I := I + Bad_Shift(Haystack(I + Needle'Length));
      -- Shift
    end loop;

    return 0;
  end QS;
```

I also added a loop variant to ensure that the while-loop will terminate. For-loops always terminate in SPARK because the loop index cannot be assigned by the user (contrary to what C allows), but while-loops or plain-loops might not terminate, hence the use of a loop variant to verify their termination.

The code presented in this post is available on GitHub: spec [6] and body [7]. Now a challenge for Frama-C users is to translate back the functional proof of QS in SPARK into C and Frama-C!

The project SPARK-by-Example [8] by Christophe Garion and Jérôme Hugues contains other examples of functionally proven string algorithms, which correspond to the SPARK version of the work done by Jens Gerlach with Frama-C in the ACSL-by-Example [9] project.

[6] https://github.com/AdaCore/spark2014/blob/master/testsuite/gnatprove/tests/string_search/string_search.ads

[7] https://github.com/AdaCore/spark2014/blob/master/testsuite/gnatprove/tests/string_search/string_search.adb

[8] https://github.com/yoogx/spark_examples/tree/master/spark-by-example

[9] https://github.com/fraunhoferfokus/acsl-by-example

National Ada Organizations

Ada-Belgium

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

Ada in Denmark

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

Ada-Deutschland

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

Ada-France

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

Ada-Spain

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

Ada-Switzerland

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