ADA USER JOURNAL

Volume 36
Number 3
September 2015

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Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.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

As the reader is aware, the 2015 volume of the Ada User Journal is celebrating the bicentennial of Ada Lovelace with a set of non-technical articles about Ada herself and her relation with Babbage. In this issue we will make an exception, and celebrate the bicentennial with one technical article, which was written in 1843: the notes to the English translation of the description of Babbage's Analytical Engine. The notes, and their creation, have since then been the subject of several analysis; this issue provides them for the analysis of the reader.

The 1843 notes article provides the description of the analytical engine, which is an interesting read, the notes with Ada's explanations of the workings of the engine (actually describing what a computer does) and Ada's reasoning of what the engine could eventually be. Can you imagine 170 years ago projecting that it could create music? I particularly encourage the reader to take a journey back in time, and attentively read the writings of Ada.

I would like to thank John Fuegi and Jo Francis for pointing out the sources we have used to reproduce this article, as well as Patricia López for the huge effort in editing them in the Journal.

In parallel to the celebration articles, the Journal continues the task of promoting and disseminating technical articles concerning the Ada language in particular and reliable software in general. The issue follows with an article worst-case execution time analysis of DO-178 Level A software, by a group of authors from Rolls-Royce Controls & Data Services and Rapita Systems, from the UK. This article concludes the publication of the Advances on Methods session of Ada-Europe 2015 (previous articles of the session were published in the March 2015 issue).

Afterwards, it provides the best papers from the Architecture Centric Virtual Integration Workshop, also an event associated with the Ada-Europe 2015 conference. In the first article, authors from the University of York, UK, explain how model weaving allows for modelling correspondences between models and how to apply it for assurance cases. Then, Peter Feiler, from the Software Engineering Institute, USA, describes an approach for the specification of verifiable requirements and for system safety analysis.

In parallel to the technical contents of the Journal, I would like also to note the featured forthcoming events.

Although not first in chronological order, and bracketed by two technical events, I would like to start with the Ada Lovelace Symposium, which will take place at the Mathematics Institute, University of Oxford on 9th and 10th December 2015. The symposium will be one of the main bicentennial celebrating events, featuring reputed speakers (among them John Barnes), discussing Ada's life and work.

Going back to chronological order, first we have the announcement of the 2nd UK conference on High Integrity Software taking place in Bristol, UK, on November 5, 2015, an event about challenges and solutions in the domain of trustworthy software engineering. Then the announcement of the 2016 International Real-Time Ada Workshop, which will take place in Benicássim, near Valencia, Spain, April 11-13 2016. An important event for the advancement of Ada technology and use in one very important domain.

To conclude, also a special note to the Ada-Europe 2016 conference, which will take place in Pisa, Italy, in the week of 13-17 June, 2016. The opportunity for Ada and Reliable Software practitioners and enthusiasts to present their work and for the community to connect in an enjoyable scenario. Recognizing the importance of parallelism, and its impact on future reliable systems, the conference includes a Special Session on Safe, Predictable Parallel Software Technologies.

The Ada-Europe conference is an outcome of contributions from the community; I encourage, and insist in asking for, your contribution!

Luís Miguel Pinho
Porto
September 2015
Email: AUJ_Editor@Ada-Europe.org

Quarterly News Digest

Jacob Sparre Andersen

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

Contents

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]

Mascot Competition Result

From: David Botton <david@botton.com>
Date: Tue, 5 May 2015 12:25:54 -0700
Subject: New Ada Mascot Image - Lady Ada with Lady Fairy
Newsgroups: comp.lang.ada

Here is the new image I hired Leah to create with the Ada Mascot and Lady Ada together :)

http://getadanow.com/img/lady_fairy_sm.png

I have also added the new image to the Mascot Store (no profits generated):

http://www.cafepress.com/adamascot

Mascot files:

http://getadanow.com/mascot.html

[See also “Mascot Competition Result”, AUJ 36-2, p. 60. —sparre]

Ada-Belgium Spring Event

From: Dirk Craeynest <dirk@cs.kuleuven.be>
Date: Tue, 2 Jun 2015 21:02:24 +0000
Subject: Ada-Belgium Spring 2015 Event, Sat 13 June 2015
Newsgroups: comp.lang.ada, fr.comp.lang.ada, be.comp.programming

---

Ada-Belgium Spring 2015 Event

Saturday, June 13, 2015, 12:00-19:00

Leuven, Belgium

including at 15:00

2015 Ada-Belgium General Assembly

and at 16:00

Ada Round-Table Discussion

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

Announcement

The next Ada-Belgium event will take place on Saturday, June 13, 2015 in Leuven.

For the 8th year in a row, Ada-Belgium decided to organize their "Spring Event", which starts at noon, runs until 7pm, and includes an informal lunch, the 22nd General Assembly of the organization, and a round-table discussion on Ada-related topics the participants would like to bring up.

Schedule

- 12:00 welcome and getting started (please be there!)
- 12:15 informal lunch
- 15:00 Ada-Belgium General Assembly
- 16:00 Ada round-table + informal discussions
- 19:00 end

Participation

Everyone interested (members and non-members alike) is welcome at any or all parts of this event.

For practical reasons registration is required. If you would like to attend, please send an email before Wednesday, June 10, 21:00, to Dirk Craeynest <Dirk.Craeynest@cs.kuleuven.be> with the subject "Ada-Belgium Spring 2015 Event", so you can get precise directions to the place of the meeting. Even if you already responded to the preliminary announcement, please reconfirm your participation ASAP.

If you are interested to become a new member, please register by filling out the 2015 membership application form[1] and by paying the appropriate fee before the General Assembly. After payment you will receive a receipt from our treasurer and you are considered a member of the organization for the year 2015 with all member benefits[2]. Early renewal ensures you receive the full Ada-Belgium membership benefits (including the Ada-Europe indirect membership benefits package).

As mentioned at earlier occasions, we have a limited stock of documentation sets and Ada related CD-ROMs that were distributed at previous events, as well as back issues of the Ada User Journal[3]. These will be available on a first-come first-serve basis at the General Assembly for current and new members. (Please indicate in the above-mentioned registration e-mail that you're interested, so we can bring enough copies.)

[1] http://www.cs.kuleuven.be/~dirk/ada-belgium/forms/member-form15.html

[2] http://www.cs.kuleuven.be/~dirk/ada-belgium/member-benefit.html

[3] http://www.ada-europe.org/auj/home/

Informal lunch

The organization will provide food and beverage to all Ada-Belgium members. Non-members who want to participate at the lunch are also welcome: they can choose to join the organization or pay the sum of 15 Euros per person to the Treasurer of the organization.

General Assembly

All Ada-Belgium members have a vote at the General Assembly, can add items to the agenda, and can be a candidate for a position on the Board[4]. See the separate official convocation for all details.

[4] http://www.cs.kuleuven.be/~dirk/ada-belgium/board/

[5] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/15/150613-abga-conv.html

Ada Round-Table Discussion

As in recent years, we plan to keep the technical part of the Spring event informal as well. We will have a round-table discussion on Ada-related topics the participants would like to bring up. We invite everyone to briefly mention how they are using Ada in their work or non-work environment, and/or what kind of Ada-related activities they would like to embark on. We hope this might spark some concrete ideas for new activities and collaborations.

Directions

To permit this more interactive and social format, the event takes place at private premises in Leuven. As instructed above, please inform us by e-mail if you would like to attend, and we'll provide you precise directions to the place of the meeting. Obviously, the number of

participants we can accommodate is not unlimited, so don't delay...

Looking forward to meet many of you!

Dirk Craeynest, President Ada-Belgium

Dirk.Craeynest@cs.kuleuven.be

Acknowledgements

We would like to thank our sponsors for their continued support of our activities: AdaCore, Barco, Katholieke Universiteit Leuven (KU Leuven), and Université Libre de Bruxelles (U.L.B.).

If you would also like to support Ada-Belgium, find out about the extra Ada-Belgium sponsorship benefits:

http://www.cs.kuleuven.be/~dirk/ada-belgium/member-benefit.html#sponsor

---

[I've heard from Dirk that they had a very pleasant afternoon. —sparre]

Survey: Programmers' Impressions of Ada

From: Edward R. Fish <onewingedshark@gmail.com>
Date: Mon, 27 Jul 2015 18:16:54 -0700
Subject: Survey of Programmers' Impressions of the Ada Language
Newsgroups: comp.lang.ada

https://docs.google.com/forms/d/15mmK_qV8P9DKEhHUnIb1RFElQFTjVTZHH0S6pO1iWrQ/viewform?usp=send_form

I posted this on reddit, but thought I'd be thorough and post it here as well -- although I certainly want input from people who aren't versed in Ada.

Ada-related Resources

Join Ada Now

From: David Botton <david@botton.com>
Date: Wed, 24 Jun 2015 15:18:45 -0700
Subject: JoinAdaNow.com and the Ada Mascot in the Wild Project
Newsgroups: comp.lang.ada

I have started a new site in the AdaNow series - http://joinadanow.com

Do you have an active Ada project and looking for developers? Looking to start one? Let me know and I'll list it under Join a Group

In addition the Ada Mascot original images for use and modification and a new project "Ada Mascot in the Wild" for Ada Mascot Sightings is hosted there:

http://joinadanow.com/#mascot

Are you using the Ada Mascot (it is completely free for any Ada use)? Have you seen it in use somewhere let me know so it can be listed.

Ada on Social Media

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Mon Aug 24 2015
Subject: Ada on Social Media

Ada groups on various social media:

- LinkedIn[1]: 2_305 members

- Reddit[2]: 820 readers

- Google+[3]: 522 members

- StackOverflow[4]: 291 followers

- Twitter[5]: 4 tweeters

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

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

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

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

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

[See also “Ada on Social Media”, AUJ 36-2, p. 62. —sparre]

Repositories of Open Source Software

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Mon Aug 24 2015
Subject: Repositories of Open Source software

GitHub: 835 repositories [1]
257 developers [1]

Rosetta Code: 620 examples [2]
30 developers [3]

Sourceforge: 245 repositories [4]

BlackDuck OpenHUB: 211 projects [5]

Bitbucket: 110 repositories [6]
17 developers [6]

OpenDO Forge: 23 projects [7]
445 developers [7]

Codelabs: 20+ repositories [8]

AdaForge: 8 repositories [9]

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

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

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

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

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

[6] http://edb.jacob-sparre.dk/Ada/on_bitbucket

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

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

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

[See also “Repositories of Open Source Software”, AUJ 36-2, p. 62. —sparre]

Ada-related Tools

OpenToken

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Tue, 12 May 2015 08:09:12 -0500
Subject: opentoken 6.0b released
Newsgroups: comp.lang.ada

opentoken 6.0b is available at

http://stephe-leake.org/ada/opentoken.html

This fixes the bugs reported since 6.0a was released.

[See also “OpenToken”, AUJ 36-2, p. 65. —sparre]

Sending E-Mail

From: Björn Lundin <b.f.lundin@gmail.com>
Date: Fri, 15 May 2015 23:21:48 +0200
Subject: Re: Email from and Ada program
Newsgroups: comp.lang.ada

> [...]

Look at AWS. It has an SMTP package.

Use as:

```
SMTP.Client.Send (
        Server => SMTP_Server,
        From  => SMTP.E_Mail ("Sender
         Name", Sender_Email_Address),
        To => Receivers,
        Subject => Subject,
        Message => Msg,
        Status  => Status);
```

Mathpaqs

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Tue, 2 Jun 2015 05:27:46 -0700
Subject: Mathpaqs release 30-May-2015
Newsgroups: comp.lang.ada

Two additions in the latest release:

- Formulas, a generic formula package with parsing, evaluation and simplification

- Contours, a generic contour plot package

[See also “Excel Writer, GNAVI, Mathpaqs and Zip-Ada”, AUJ 34-4, p. 200. —sparre]

Multiprecision Integers

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Wed, 03 Jun 2015 11:09:17 -0700
Subject: Re: Mathpaqs release 30-May-2015
Newsgroups: comp.lang.ada

> I am really interested by your multiprecision integers. I am happy to see that they are implemented in Ada.

The trouble with his multiprecision integers is that, for each variable, you have to specify the number of "digits" it can hold. You can sometimes get around this by using the result of an expression as the initialization of an object. But sometimes it is difficult, especially if you want to write portable code, since the size of a "digit" is determined by a constant from System.

The alternative to this is to have unbounded integers, with the number of "digits" determined by the value. This is generally implemented using access types, allocation, and deallocation, which can be difficult to get right.

However, in the beta version of the PragmAda Reusable Components for ISO/IEC 8652:2007, there is an Unbounded_Integers pkg that does not (explicitly) use access types. The division algorithm might be a bit slow.

> I used to write a binding to GMP and MPFR but I only considered "Unbounded Integers", implemented as controlled types, just like unbounded_strings are implemented. Do you know how fast is your Library compared to GMP?

Advantages to a binding to something like GMP is that it is well tested and fast. The disadvantage is that it is a binding to something written in not-Ada. Also, a decent binding might use 3 subprogram calls where a native pkg would only use 1 (your code calls the thick binding, which calls the thin binding, which calls the imported code), and convert types, which might be enough to make up for the speed difference. The PragmARCs are at

https://pragmada.x10hosting.com/pragmarc.htm

Mosquitto

From: Per Sandberg <per.s.sandberg@bahnhof.se>
Date: Mon, 08 Jun 2015 20:10:37 +0200
Subject: ANN: mosquitto-ada-1.0.0
Newsgroups: comp.lang.ada

mosquitto-ada is an Ada-interface to the MQTT it depends on the mosquitto client library.

Website: https://github.com/persan/mosquitto-ada/Per

[See also “Mosquitto”, AUJ 36-2, p. 65. —sparre]

Simple Components

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 14 Jun 2015 10:38:29 +0200
Subject: ANN: Simple Components v4.7
Newsgroups: comp.lang.ada

The new version provides implementations of HTTP, HTTPS and MODBUS clients. The intended use is for several clients to share single Ada task, which may be important for embedded and heavy-duty targets. Traditional synchronous operating mode is supported as well.

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

[See also “Simple Components”, AUJ 36-1, p. 14. —sparre]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 1 Jul 2015 21:39:11 +0200
Subject: ANN: GNAT GPL 2015 updates
Newsgroups: comp.lang.ada

The following packages are updated to support GNAT GPL 2015 and GtkAda 3.8.3. Minor issues related to the new compiler version and GtkAda are fixed.

Ada industrial control widget library

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

[See also “Industrial Control Widget Library”, AUJ 35-3, p. 157. —sparre]

Simple Components for Ada

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

GtkAda contributions

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

[See also “GtkAda Contributions”, AUJ 35-3, p. 155. —sparre]

Units of measurement for Ada

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

[See also “Units of Measurement”, AUJ 35-3, p. 156. —sparre]

Fuzzy sets for Ada

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

[See also “Fuzzy Sets”, AUJ 35-3, p. 157. —sparre]

Fuzzy machine learning

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

[See also “Fuzzy machine learning framework”, AUJ 33-3, p. 143. —sparre]

P.S. Note that this compiler version introduces 64-bit Stream_Offset even if the target is 32-bit. The project scenario controlling atomic access must be set on GCC-long-offsets.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 22 Aug 2015 08:39:21 +0200
Subject: ANN: Simple Components v4.9 released
Newsgroups: comp.lang.ada

The new version provides an implementation of ELV/eQ-3 MAX! cube protocol.

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

GNAT

From: Anh Vo <anhvofrcaus@gmail.com>
Date: Mon, 15 Jun 2015 08:39:19 -0700
Subject: GNAT GPL 2015
Newsgroups: comp.lang.ada

I happened to check http://libre.adacore.com/. Voila, GNAT GPL 2015 is available. I am anxious to download it after getting from my current vacation.

Gnoga

From: David Botton <david@botton.com>
Date: Wed, 1 Jul 2015 07:43:49 -0700
Subject: GNOGA v1.1 Released - Ada Cloud Desktop and Mobile Development
Newsgroups: comp.lang.ada

GNOGA v1.1 for Ada is now available from http://gnoga.com or via git from http://sourceforge.net/projects/gnoga

V1.1 in addition to bug fixes and vastly increased stability adds:

- Completed multimedia bindings
- Easier boot file creation by just including boot.js in any html file
- Direct HTTPS and Secure Websockets
- HTTP polling with Ajax including fallback support (using auto.html bootfile)

[See also “Gnoga”, AUJ 36-2, p. 63. —sparre]

Qt5Ada

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Thu, 2 Jul 2015 20:51:34 -0700
Subject: Announce : Qt5Ada version 5.5.0 (432 packages) and VTKAda version 6.2.0 (656 packages) release 02/07/2015 free edition
Newsgroups: comp.lang.ada

Qt5Ada is Ada-2012 port to Qt5 framework (based on Qt 5.5.0 final).

Qt5ada version 5.5.0 open source and qt5c.dll, libqt5c.so(x32 and x64), libqt5c.dylib built with Microsoft Visual Studio 2012 in Windows, gcc x86-64 (x86-32) in Linux and Mac OSX.

Package tested with GNAT GPL Ada compiler in Windows 32bit and 64bit, Linux x86, Linux x86-64, Debian 7.3 and Mac OSX 10.8.5.

It supports GUI, SQL, multimedia, web, network, touch devices, sensors, navigation and many others things.

Added QtLocation support, new packages and demos.

Qt5Ada for Windows, Linux (Unix) and Mac (OSX) 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) or download from http://ul.to/3u8uz26c

My configuration script to build Qt 5.5 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/5.5"

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

I do this work on my own risk and 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, please let me know.

[See also “Qt5Ada”, AUJ 36-1, p. 16. —sparre]

SDLAda

From: Luke A. Guest <laguest@archeia.com>
Date: Sun, 12 Jul 2015 07:25:36 -0700
Subject: ANN: First official release of SDLAda
Newsgroups: comp.lang.ada

After much stagnation, I have finally tagged a v1.0.0 release of my SDL 2.0.3 bindings (http://libsdl.org/).

There's still a lot missing, but most of the functionality to get something working is in there.

You can get it and test it from GitHub:

https://github.com/Lucretia/sdlada/tree/v1.0.0

GNAT for Atmel SAM4S

From: Patrick Noffke <patrick.noffke@gmail.com>
Date: Tue, 28 Jul 2015 13:08:03 -0700
Subject: ANN: GNAT GPL 2015 Atmel SAM4S Ravenscar patches
Newsgroups: comp.lang.ada

A makefile and patches be found here:

https://github.com/patricknoffke/ada-mcu

I have added runtime support for various peripherals, with BSD 3-Clause license.

Gnoga as a Web Platform?

From: David Botton <david@botton.com>
Date: Thu, 06 Aug 2015 18:33:36 +0000
Subject: Gnoga as a Web platform
Newsgroups: gmane.comp.lang.ada.gnoga

While I am continuing towards more development of Gnoga for "native" use on local devices. It is of course an excellent platform for web development.

I think that it affords us an interesting opportunity for Ada. To provide a full contact management / web site environment in a compiled application. This of course is more secure than using typical scripting based systems like Drupal, Joomla, Wordpress, etc. They are often hacked and easily modified once in. Is anyone interested in being part of a project to create a CMS / web site system using Gnoga?

Cortex GNAT Run Time Systems

From: Simon Wright <simon@pushface.org>
Date: Sun, 09 Aug 2015 22:10:32 +0100
Subject: STM32F4 GNAT Run Time Systems project renamed
Newsgroups: comp.lang.ada

The project previously known as STM32F4 GNAT Run Time Systems is renamed to Cortex GNAT Run Time Systems, at

https://sourceforge.net/projects/cortex-gnat-rts/

I'll soon be updating with the Arduino Due version.

[See also “STM32F4 GNAT Run Time Systems”, AUJ 36-2, p. 64. —sparre]

From: Simon Wright <simon@pushface.org>
Date: Wed, 12 Aug 2015 12:07:06 +0100
Subject: ANN: Cortex GNAT Run Time Systems release 20150810
Newsgroups: comp.lang.ada

Now available, for STM32F429I-DISCO and Arduino Due. No change to RTS facilities in this release.

https://sourceforge.net/projects/cortex-gnat-rts/files/20150810/

Choosing a GUI Library

From: Trish Cayetano <trishacayetano@gmail.com>
Date: Mon, 10 Aug 2015 03:24:35 -0700
Subject: GUI for Ada (GPS with GtkAda or GtkGlade GUI Builder)
Newsgroups: comp.lang.ada

I am done with the functionality of my Ada program (using GPS) and next is to make it pretty by having a GUI instead of a text based.

Please advise what shall I use to build the GUI...

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 10 Aug 2015 14:17:01 +0200
Subject: Re: GUI for Ada (GPS with GtkAda or GtkGlade GUI Builder)
Newsgroups: comp.lang.ada

> [...]

Two major contenders are GTK and Qt.

GTK has the problem that the developer team does everything possible to keep in backward incompatible and going on removing functionalities without any replacement. Another problem is that GtkAda is kept well behind, being released only once in a year. Binary distribution of GTK for Windows is almost not maintained. Building it from sources under Windows practically impossible so presently you have to keep it working form 3.8 and 3.10 which (see above) is not trivial at best. The main advantage is that GtkAda is AdaCore.

Qt has the problem of multiple Ada bindings of uncertain quality and maintenance. I didn't use Qt so I cannot say anything regarding Qt itself.

To put things clear:

GPS is an IDE designed in GtkAda. It does not limit you to use any other GUI framework or same version of GTK (GPS is 3.8.2, I believe) Actual GTK for GtkAda is 3.8.3. Actual GTK is 3.10 or higher. The latest official binary GTK for Windows is 3.6.something.

GLADE is a GUI builder for Gtk. There are different opinions on it, mine is (I am doing a lot of stuff in GtkAda) never touch it if you want to design something beyond simple input forms.

From: Maciej Sobczak <maciej@msobczak.com>
Date: Mon, 10 Aug 2015 07:36:18 -0700
Subject: Re: GUI for Ada (GPS with GtkAda or GtkGlade GUI Builder)
Newsgroups: comp.lang.ada

> [...]

Considering the fact that the functionality is already done and therefore it will *not* be put in the GUI layer, you are free to choose the technology that is optimal for GUI. This need not be the same that you have used to implement the functional parts.

I'm playing the devil's advocate now, but really - different languages have different strengths and while eye-candy-oriented languages are not optimal for implementing critical functionality, Ada is not the sharpest knife in the GUI drawer, either.

Personally, I would limit my choices to HTML vs. C++/Qt, depending on what this GUI is going to do. Both options are known to be portable and offer modern or even spectacular results with a lot of know-how ready to be reused from the web.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 13 Aug 2015 13:59:46 +0200
Subject: Re: GUI for Ada (GPS with GtkAda or GtkGlade GUI Builder)
Newsgroups: comp.lang.ada

> Does it make sense to write application's "business-logic" (back-end) in higher-level and type-safe language like Ada and then write front-end in e.g. PyQt and call Ada code in the form of Python's extension module?

No. Because GUI code is usually bigger and messier than the application code. It is more fragile and requires much more maintenance. Furthermore it is the only thing the customer sees and actually pays attention.

> Is it "best of both worlds" - having logic written in type-safe language and GUI in productive environment like e.g. PyQt or it is actually "the worst of the two" by losing type-safety since extension module should use C convention and possibly one will also lose advantage of using Python?

No. Maintenance costs are 10 times of developing costs. For GUI code it is even more than that, because once deployed, the customer starts requiring modifications. Whatever mythical productivity it does not matter in the end anyway.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 13 Aug 2015 14:50:12 +0200
Subject: Re: GUI for Ada (GPS with GtkAda or GtkGlade GUI Builder)
Newsgroups: comp.lang.ada

> [...] So, what is the solution?

>

> Writing everything in Python :-)

[...]

> Or, do you suggest writing everything in e.g. C++?

I suggest writing everything in Ada.

From: David Botton <david@botton.com>
Date: Wed, 19 Aug 2015 07:36:10 -0700
Subject: Re: GUI for Ada (GPS with GtkAda or GtkGlade GUI Builder)
Newsgroups: comp.lang.ada

> GUI's with Ada are my biggest complaint with Ada. There seems to be no good, easy to use options.

I can only assume you mean GUI builders based on the rest of your post. Ada has (sadly license encumbered of all things on the Ada side...) versions of Gtk and Qt (same for any platform) and has GWindows which is superior to any Windows framework I've used on any platform and Gnoga which is already top notch for web apps and not bad for Desktop UIs but improving daily.

> There are several other options that are out there (GNOGA? GNAVI? some others) they seem like much more "demo" than anything else.

Gnavi the Ada Delphi clone sits on top of GWindows, in theory if someone wanted could get Gnavi up and running again, was fully functional but I never packaged it. GWindows has been used for countless professional projects and looks as good as any windows app will and is easier to develop in then other frameworks. So as I said I assume you are thinking GUI dev tools not frameworks.

> I could never get it to work properly at all, clearly not polished, at the very least, not documented well enough.

Not sure which you are talking about, but there is descent community support for GWindows and Gnoga on their lists. If you use the Sourceforge versions of either you will have no issues getting them to work and both are very polished in terms of build and have decent examples and tutorials. Of course you could contribute funds for a pro documentation writer for less than most pay tools cost and get exactly what you ask for :)

> Lazarus for FreePascal is very nice

Sadly, Gnavi was already complete before Lazarus started and could easily be way beyond it today, but there was a long period when there was no real true free Ada compiler and I had already stopped using native Windows as a dev platform.

> I guess there are not enough Ada experts out there that care enough to have free GUI tools...

Not just those that did got turned off by licensing issues when it would have made a difference. Of course today FSF GNAT is in good shape and so some like myself have returned and started work on new tools like Gnoga in the hopes that FSF GNAT and more community supported tools will come along since corporate visions are too short sited to value community and its contributions.

> Yes I do embedded work, so I don't always need a GUI. But I create stuff for my colleagues & customers as well and nobody will put up with a CLI application nowadays.

You would find Gnoga ideal in that situation. You can easily use existing HTML layout tools (I've posted some examples using web based ones on the Gnoga list) and in a few lines of code have that up and useful.

>> Please advise what shall I use to build the GUI...

Today I would only recommend Gnoga because of the flexibility for remote GUI use, cross platform completely to desktop, mobile and cloud.

If your app needed intense GUI use on desktop can even combine GTK (for the intense real time graphics) and Gnoga (forms, general use etc). (See the Gtk native doc in the docs dir).

Command Line Parser Generator

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Thu, 20 Aug 2015 15:15:33 +0200
Subject: ANN: Command Line Parser Generator
Newsgroups: comp.lang.ada

I would like to announce that I have brought my command line parser generator[1] to (what I assume is) a usable state.

Instead of manually implementing command line parsing using some library, you can collect all the ways your program can be called as procedures in a package.

A simple example:

```
package Filter is
   procedure Show_Help (Help : Boolean);
   -- Shows the usage instructions (no
   -- matter the value of 'Help').

   procedure Process (
         Source_File : String := "";
         Target_File : String := "");
   -- Empty file names are mapped to
   -- respectively 'Standard_Input'
   -- and 'Standard_Output'.
end Filter;
```

Running (yes, I need a shorter name for it):

```
command_line_parser_generator-run Filter
```

in the directory containing 'filter.ads' generates 'generated/filter-driver.adb', which when compiled transforms command line arguments into procedure calls like this:

```
--help  -> Filter.Show_Help (Help => True);
--source_file=data  -> Filter.Process
         (Source_File => "data");
--target_file=out  -> Filter.Process
         (Target_File => "out");
--source_file=data --target_file=out
   -> Filter.Process (Source_File => "data",
                      Target_File => "out");
```

There are currently two known issues with the tool [2]:

a) The tool assumes that all non-String types have a 'Value attribute, without checking if this is the case.

b) The tool doesn't check if the type of a formal parameter has a primitive Value function (which should override the 'Value attribute).

Enjoy!

[1] http://repositories.jacob-sparre.dk/command-line-parser-generator/wiki/Home

[2] http://repositories.jacob-sparre.dk/command-line-parser-generator/issues?status=new&status=open

GtkAda Contributions

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 22 Aug 2015 22:14:03 +0200
Subject: ANN: GtkAda contributions v3.13 released
Newsgroups: comp.lang.ada

The library deals with the following issues:

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

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

The release is focused on working around incompatibilities introduced with newer versions of GTK.

- Added workaround get/set string operations for tree model, store and list store (Gtk.Missed). The standard versions have critical bug;
- Messages filter added to Gtk.Main.Router.GNAT_Stack;
- Add_Button_From_Stock added to Gtk.Missed to work around Gtk.Dialog.Add_Button. (Add_Button generates warnings about GtkButton:use-stock being deprecated);
- Add_Named and Add_Stock_Attribute added to Gtk.Messed to work around deprecated "stock-id" property of GtkCellRendererPixbuf;
- Say procedures in Gtk.Main.Router and in Gtk.Main.Router.GNAT_Stack changed in order to be independent on GtkAda.Dialogs (for the same reason);
- Bug fixes.

[See also “GtkAda Contributions”, AUJ 35-3, p. 155. —sparre]

Ada-related Products

CodePeer

From: AdaCore Press Center
Date: Wed May 6 2015
Subject: AdaCore Releases CodePeer 3.0
URL: http://www.adacore.com/press/codepeer-3-0/

Advanced static analysis tool for Ada has been qualified under DO-178B and EN50128, adds support for IEEE 754 floating point semantics and enhances support for project files

Boston, Mass. – Embedded Systems Conference, NEW YORK and PARIS, May 6, 2015 – AdaCore, a leading provider of development and verification tools for critical software, today released CodePeer 3.0, a major new version of its advanced static analysis tool for the automated review and validation of Ada source code. CodePeer 3.0 includes a variety of enhancements that help developers detect potential run-time and logic errors early in the software life cycle, and its deep analysis can directly support formal certification against industry-specific safety standards.

Among the new benefits of CodePeer 3.0, tool qualification material for both the avionics and railway domains is available as a product option.

“With CodePeer 3.0, our customers can take advantage of the tool’s more robust capabilities for automated code review,” said Arnaud Charlet, CodePeer Product Manager at AdaCore. “By meeting stringent industry standards for tool usage in the aviation and rail industries, CodePeer has a proven track record in the most demanding systems and can help customers in any application domain. The tool simplifies the verification effort by detecting subtle bugs in both new code that is being developed, and in existing code bases that need to be analyzed for vulnerabilities.”

For avionics applications CodePeer has been qualified as a Software Verification Tool under DO-178B, a standard that is required by certification authorities such as the FAA in the U.S. In particular, CodePeer automates a number of verification activities defined in paragraph 6.3.4f (“Accuracy and consistency”) of the DO-178B standard. These activities include detecting errors such as values outside the bounds of an Ada type or subtype, buffer overflows, integer overflow or wraparound, division by zero, use of uninitialized variables, and floating point underflow. The DO-178B qualification material available as an option with CodePeer 3.0 demonstrates that the tool performs these activities.

CodePeer has also been qualified for EN50128, the highest international standard for safety integrity concerning software for railway control and protection, including communications, signaling and processing systems. The EN50128 qualification material addresses the following:

- Boundary value analysis to detect attempts to dereference a pointer that could be null, values outside the bounds of an Ada type or subtype, buffer overflows, integer overflow or wraparound, and division by zero.
- Control flow analysis to detect suspicious and potentially incorrect control flows, such as unreachable code, redundant conditionals, loops that either run forever or fail to terminate normally, and subprograms that never return.
- Data flow analysis to detect suspicious and potentially incorrect data flows, such as variables that are read before they are written (uninitialized variables), variables written more than once without being read (redundant assignments), variables that are written but never read, and parameters with an incorrect mode (unread “in” parameter, unassigned “out” parameter).

CodePeer 3.0 also adds many new features, including support for precise IEEE 754 floating point semantics, added flexibility in analyzing complex projects, improved support for legacy Ada compilers, more precise diagnostic messages, and a new check on parameter aliasing.

CodePeer is fully integrated into Adacore’s GNAT Pro development environment and comes with a number of complementary static analysis tools common to the technology – a coding standard verification tool (GNATcheck), a source code metric generator (GNATmetric) and a document generator.

A demo of the tool highlighting the new features introduced in CodePeer 3.0 will be available soon; for a demo of the previous version of the product please visit http://www.adacore.com/knowledge/demos/codepeer-2-3/

[...]

[See also “CodePeer”, AUJ 35-1, p. 10. —sparre]

VectorCAST Integration with CodePeer

From: Vector Software
Date: Tue Jul 14 2015
Subject: Vector Software Announces Integration with AdaCore's CodePeer 3.0 Static Analysis Tool
URL: https://www.vectorcast.com/news/vector-software-press-releases/2015/vector-software-announces-integration-adacores-codepeer-30

Latest VectorCAST integration provides Ada developers with powerful tools for automated code review and validation

July 14, 2015

Providence, RI – 7/14/2015 - Vector Software, the world's leading provider of innovative software solutions for robust embedded software quality, announced today an integration of the VectorCAST test automation platform with CodePeer 3.0 - AdaCore's advanced static code analysis tool for Ada, including version 2012.

VectorCAST and CodePeer now provide Development and QA teams with the ability to focus test efforts in areas most susceptible to errors. An additional capability allows developers of legacy applications the ability to augment code covered during unit/integration and system test with code considered "clean" by CodePeer. Clean code can be imported into the VectorCAST/CBA (Covered by Analysis) facility to increase coverage levels. Code considered "not clean" would be designated for additional testing with VectorCAST.

AdaCore's CodePeer 3.0 Advanced Static Analysis tool detects possible run-time errors including: IEEE 754 Floating Point semantics, buffer overflows, integer overflow or wraparound, division by zero, index/range checks, uninitialized variables, unused assignments, redundant and invariant constructs, infinite loops, race conditions, and suspicious implicit contracts in source code.

This latest integration benefits all customers working with Ada but has some additional certification advantages for those working protection and control systems for Avionics or Railway, where software quality and certification are mandated such as: RTCA D-178B/C, EUROCAE ED-12B/C, or CENELEC, EN 50128.

"Providing combined views of the static and dynamic analysis results offers novel capabilities in terms of efficient verification for high-assurance systems", said Cyrille Comar, AdaCore President. "The VectorCAST environment allows our customers to get the best of CodePeer static analysis by helping them to concentrate on the parts of the application that are less well covered by dynamic tests."

"With AdaCore's CodePeer 3.0 advanced static analysis and the VectorCAST Test Automation Platform's newly engineered features, such as Covered by Analysis, we are able to provide new and legacy Ada developed projects with tools to focus test efforts in areas that will provide the best return on investment", said William McCaffrey, Chief Operating Officer at Vector Software.

[...]

GNAT Pro for VxWorks

From: AdaCore Press Center
Date: Tue Jul 21 2015
Subject: AdaCore's GNAT Pro Available for Wind River VxWorks 7
URL: http://www.adacore.com/press/gnat-pro-available-for-wind-river-vxworks-7/

Offers Full Ada, Support for the Latest Wind River RTOS, More Seamless Integration with Workbench

SANTA CLARA, CA – Embedded Systems Conference, NEW YORK and PARIS, July 21, 2015 – AdaCore today announced the continuing extension of its Wind River® VxWorks® real-time operating system (RTOS) support, with the implementation of the GNAT Pro development environment on VxWorks 7.

AdaCore engineers worked closely with Wind River on this new product, ensuring that it would support both single- and multi-core systems, as well as other architectures. Enhancements over previous versions include a completely reengineered open source debugger protocol and more seamless integration with Wind River Workbench, and the development environment handles both all-Ada and multi-language applications.

"Wind River provides proven, reliable and stable solutions," said Jerome Guitton, AdaCore's VxWorks product manager. "With VxWorks 7, the company has elevated its technology to entirely new heights, moving to a much broader integration of embedded solutions and big data. These are important attributes in helping us provide market-leading solutions for our joint customers, and the AdaCore plug-in for Wind River Workbench will dramatically improve their experience."

GNAT Pro for VxWorks 7 offers a variety of benefits:

- Implementation of all editions of the Ada language standard, including the latest version Ada 2012
- Support for VxWorks 7 kernel modules and real-time processes
- Continued support for PowerPC, Intel and ARM instruction sets
- Mixed-language support, allowing applications consisting of Ada, C and C++
- SMP support
- Extensive GNAT library
- Ada unit testing framework (AUnit)
- Dependable "front-line" support from AdaCore

"AdaCore's GNAT Pro is well established among users of Wind River platforms, especially in the aerospace and defense market," said Prashant Dubal, director of VxWorks product management at Wind River. "This new version of GNAT Pro for VxWorks 7 is the latest step in the long and successful strategic partnership between AdaCore and Wind River."

[...]

Ada and Operating Systems

Mac OS X: GNAT for ARM-EABI

From: Simon Wright <simon@pushface.org>
Date: Sun, 21 Jun 2015 17:25:51 +0100
Subject: ANN: GNAT GPL 2015 arm-eabi for Darwin
Newsgroups: comp.lang.ada

Released at

https://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2015-arm-eabi-darwin-bin/

README:

This is GNAT GPL 2015, rebuilt as a cross-compiler from Mac OS X to arm-eabi. Two runtimes (full Ravenscar, and small-footprint Ravenscar, respectively ravenscar-full-stm32f4 and ravenscar-sfp-stm32f4) are supplied, configurable for three STM32F boards (STM32F4-DISCO, STM32F429-DISCO, and STM32F7-EVAL). Examples are included.

The compiler is known to run on Mavericks and Yosemite.

For installation, untar gnat-gpl-2015-arm-eabi-darwin-bin.tar.bz2, enter gnat-gpl-2015-arm-eabi-darwin-bin/ (there is a README) and run doinstall (sudo doinstall). Note that you must have a working host compiler (the official GNAT GPL 2015 from [1]), and this compiler must be installed on top of it.

Additionally, stlink-darwin-bin.zip contains a .tar.gz file with the stlink utilities used to communicate with the boards over USB, and a README which details installation.

Usage notes are in the AdaCore "GNAT Pro User's Guide Supplement for Cross Platforms"[2], specifically in section K.2[3].

[1] http://libre.adacore.com

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

[3] http://docs.adacore.com/gnat_ugx-docs/html/gnat_ugx_14.html#SEC204

[See also “Mac OS X: GCC for ARM-EABI”, AUJ 36-2, p. 68. —sparre]

From: Simon Wright <simon@pushface.org>
Date: Thu, 06 Aug 2015 18:19:03 +0100
Subject: New Mac OS X GNAT/GCC arm-eabi compiler releases
Newsgroups: comp.lang.ada

I've rebuilt both GCC 5.1.0 and GNAT GPL 2015 for arm-eabi to support Cortex-M3 as in the Arduino Due, Cortex-M4, and Cortex-M4F as in the STM32F4 boards.

GCC 5.1.0 at

https://sourceforge.net/projects/gnuada/files/GNAT_GCC%20Mac%20OS%20X/5.1.0/arm-eabi-bis/

GNAT GPL 2015 at

https://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2015-arm-eabi-darwin-bin-bis/

Fedora on ARM: Gprbuild

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 28 Jun 2015 20:09:05 +0200
Subject: Re: gprbuild fun
Newsgroups: comp.lang.ada

> [...]

Under Fedora ARM gprbuild is broken (wrong target). The fix is to rename it to /usr/bin/gprbuild.old and use

```
#!/bin/sh
gprbuild.old --target=armv7hl-redhat-linux-gnueabi $*
```

instead. It is a pity that gnatmake is going to be ditched.

Mac OS X: GCC

From: Simon Wright <simon@pushface.org>
Date: Mon, 29 Jun 2015 16:40:42 +0100
Subject: ANN: GCC 5.1.0 for Mac OS X with GNAT GPL 2015 tools
Newsgroups: comp.lang.ada

See

https://sourceforge.net/projects/gnuada/files/GNAT_GCC%20Mac%20OS%20X/5.1.0/

(in the native-2015 directory).

Much as before! Changes:

- Tools from GNAT GPL 2015.

- Should be possible to install anywhere you prefer.

[See also “Mac OS X: GCC”, AUJ 36-2, p. 68. —sparre]

Windows: GNAT

From: David Botton <david@botton.com>
Date: Mon, 29 Jun 2015 12:36:50 -0700
Subject: 32 and 64 bit Gnat for Windows updated
Newsgroups: comp.lang.ada

The FSF GNAT distro I use for Windows testing TDM-GCC has been updated to 5.1.0

http://tdm-gcc.tdragon.net/

For other FSF GNAT version see - http://GetAdaNow.com

Debian and Fedora: GtkAda

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 30 Jun 2015 22:04:04 +0200
Subject: ANN: GtkAda 3.8.3 packaged
Newsgroups: comp.lang.ada

GtkAda 3.8.3 from GNAT GPL 2015 is packaged for Debian and Fedora:

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

Supported are x86 32/64-bits and ARM

[See also “Debian: GtkAda for ARMv7”, AUJ 36-2, p. 67. —sparre]

Fedora: Gprbuild, XML/Ada and AWS

From: Pavel Y. Zhukov <pavel.y.zhukov@gmail.com>
Date: Wed, 1 Jul 2015 12:03:20 -0700
Subject: ANN gprbuild/xmlada/aws 2015 in Fedora. Arm is supported
Newsgroups: comp.lang.ada

Xmlada gprbuild, aunit and aws were updated to latest 2015 release. All of them will be released with Fedora 23. Finally we've got these packages built for ARM architecture.

Archlinux

From: Rod Kay <rodakay@internode.on.net>
Date: Tue, 28 Jul 2015 07:11:18 -0700
Subject: Ann: Ada packages on Archlinux updated to GPL15.
Newsgroups: comp.lang.ada

The new packages are on AUR4. If anyone would care to try these packages then any feedback would be welcome.

Thanks again to John Marino for paving the way with the equivalent BSD packages.

From: Rod Kay <rodakay@internode.on.net>
Date: Tue, 28 Jul 2015 11:16:26 -0700
Subject: Re: Ann: Ada packages on Archlinux updated to GPL15.
Newsgroups: comp.lang.ada

[...]

https://aur4.archlinux.org/packages/ada-web-server/

https://aur4.archlinux.org/packages/ahven

https://aur4.archlinux.org/packages/asis

https://aur4.archlinux.org/packages/florist

https://aur4.archlinux.org/packages/gnat_util/

https://aur4.archlinux.org/packages/gprbuild/

https://aur4.archlinux.org/packages/gtkada/

https://aur4.archlinux.org/packages/polyorb/

https://aur4.archlinux.org/packages/prepare_gnat_util/

https://aur4.archlinux.org/packages/sphinxcontrib-adadomain/

https://aur4.archlinux.org/packages/xmlada/

GPS should only be a day or so away.

From: Rod Kay <rodakay@internode.on.net>
Date: Fri, 31 Jul 2015 01:36:46 -0700
Subject: Re: Ann: Ada packages on Archlinux updated to GPL15.
Newsgroups: comp.lang.ada

> GPS is now available.

https://aur4.archlinux.org/packages/gnat-gps

Windows 10: Gnoga

From: David Botton <david@botton.com>
Date: Sat, 1 Aug 2015 20:04:28 -0700
Subject: Gnoga on Windows 10
Newsgroups: comp.lang.ada

I've tested windows 10 using TDM-GCC 5.1 (64 bit windows) and msysgit shell as well as the built in command prompt, in both cases all built and ran properly (less the sql examples and python 2.7 examples since I don't have build for those windows lib around). In both cases build time was much faster on Windows 10 then 8 (using same gcc version) and in fact was outpacing the linux builds on the same machine for the first time ever.

My tests ran without issue.

Of course Gnoga runs without issue on any platform that supports FSF GNAT 4.7 (and also has run time support for gnat sockets) and above including Raspberry Pi, Linux, *BSD, Windows 32 and 64 bits, Mac OSX, etc.

From: David Botton <david@botton.com>
Date: Sat, 1 Aug 2015 22:12:55 -0700
Subject: Re: Gnoga on Windows 10
Newsgroups: comp.lang.ada

I've modified the Gnoga Makefile to now build a static libsqlite3.a as part of its build, so now that is available with Windows as well out of the box and works well.

References to Publications

Shared Resource Design Patterns

From: Jim Rogers
Date: Mon May 25 2015
Subject: Shared Resource Design Patterns
URL: http://sworthodoxy.blogspot.dk/2015/05/shared-resource-design-patterns.html

Summary

Many applications are constructed of groups of cooperating threads of execution. Historically this has frequently been accomplished by creating a group of cooperating processes. Those processes would cooperate by sharing data. At first, only files were used to share data. File sharing presents some interesting problems. If one process is writing to the file while another process reads from the file you will frequently encounter data corruption because the reading process may attempt to read data before the writing process has completely written the information. The solution used for this was to create file locks, so that only one process at a time could open the file. Unix introduced the concept of a Pipe, which is effectively a queue of data. One process can write to a pipe while another reads from the pipe. The operating system treats data in a pipe as a series of bytes. It does not let the reading process access a particular byte of data until the writing process has completed its operation on the data.

Various operating systems also introduced other mechanisms allowing processes to share data. Examples include message queues, sockets, and shared memory. There were also special features to help programmers control access to data, such as semaphores. When operating systems introduced the ability for a single process to operate multiple threads of execution, also known as lightweight threads, or just threads, they also had to provide corresponding locking mechanisms for shared data.

Experience shows that, while the variety of possible designs for shared data is quite large, there are a few very common design patterns that frequently emerge. Specifically, there are a few variations on a lock or semaphore, as well as a few variations on data buffering. This paper explores the locking and buffering design patterns for threads in the context of a monitor. Although monitors can be implemented in many languages, all examples in this paper are presented using Ada protected types. Ada protected types are a very thorough implementation of a monitor.

[...]

SPARK 2014 Makes Formal Verification Easier

From: Yannick Moy
Date: Mon Jun 1 2015
Subject: SPARKSkein: From tour-de-force to run-of-the-mill Formal Verification
URL: http://www.spark-2014.org/entries/detail/sparkskein-from-tour-de-force-to-run-of-the-mill-formal-verification

Subject: SPARKSkein: From tour-de-force to run-of-the-mill Formal Verification

From: Yannick Moy
Date: Mon Jun 1 2015
URL: http://www.spark-2014.org/entries/detail/sparkskein-from-tour-de-force-to-run-of-the-mill-formal-verification

In 2010, Rod Chapman, then technical leader of the SPARK team at Altran, released an implementation in SPARK of the Skein cryptographic hash algorithm. Using the previous version of the SPARK technology, Rod proved that his implementation was free of run-time errors (even found a subtle corne-case bug in the C reference implementation), but that was no trivial task, as he explained later in a paper surveying past projects in SPARK:

"The proofs of type safety turned out to be quite tricky. Firstly, finding the correct loop invariants proved difficult, and this was compounded by the plethora of modular types and non-linear arithmetic in the VC structures. Of the 367 VCs, 23 required use of the Checker to complete the proof - not bad but these still required a substantial effort to complete."

Considering that Rod is a leading expert in the technology, that assessment alone could deter non-expert users from ever attempting a similar project!

Now comes SPARK 2014 and the new version of the SPARK technology. We have recently translated the code of SPARKSkein from SPARK 2005 to SPARK 2014, and used GNATprove to prove absence of run-time errors in the translated program. The difference between the two technologies is striking. The heroic effort that Rod put in the formal verification of the initial version of SPARKSkein could now be duplicated with modest effort and modest knowledge of the technology [...]

Book: Ada and SPARK on ARM Cortex-M

From: Maciej Sobczak <maciej@msobczak.com>
Date: Sat, 13 Jun 2015 14:18:22 -0700
Subject: New book: Ada and SPARK on ARM Cortex-M
Newsgroups: comp.arch.embedded

I am pleased to announce that the tutorial titled "Ada on ARM Cortex-M", which was announced here in its early stages, evolved and finally became a regular printed book:

http://www.lulu.com/shop/maciej-sobczak/ada-and-spark-on-arm-cortex-m/paperback/product-22195745.html

The complete book content is still available on-line here:

http://inspirel.com/articles/Ada_On_Cortex.html

This book is intended as an introduction for Ada beginners and covers also the basic concepts of SPARK that allows to write programs that can be statically proven to be free from runtime errors, which a very efficient approach for embedded systems.

The Arduino Due board was used as a base for practical examples, but the book is intended to highlight the exploration process from the very fundamental basics and as such can be used with other boards and other Cortex-M microcontrollers.

[See also "Tutorial: ARM Cortex-Mx", AUJ 36-2, p. 69. —sparre]

From: Jerry Petrey <gpetrey@earthlink.net>
Date: Thu, 25 Jun 2015 16:54:30 -0700
Subject: Re: New paper book: Ada on ARM Cortex-M
Newsgroups: comp.lang.ada

[...] Nice work. I am having a lot of fun getting GNAT Ada running on a lot of ARM boards, creating support for the ARM on-chip peripherals and creating drivers for common external sensors, displays, etc. It is great that others are seeing the power of Ada on these powerful platforms. I am not that happy with the 2015 GNAT ARM release - it doesn't seem to work correctly, doesn't have some of the support it promises and has changed a lot from the initial release making a lot of work for me to port what I have already done. I think I will mostly stick with the 2014 release for now.

Video: Case Statements

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Sat Aug 1 2015
Subject: Case statements in Ada (video)
URL: http://ada.tips/case-statements-in-ada-video.html

About case statements in Ada and how they are a bit special, compared to in many other languages.

The presentation also mentions subtypes (subsets). Some Ada 2012 features are used.

http://www.jacob-sparre.dk/ada/videos/case-statements.mp4

http://www.jacob-sparre.dk/ada/videos/case-statements.ogv

[One of the submissions for the "Learn Ada Now" competition. —sparre]

Coding Standards

From: Markus Schöpflin
Date: Thu, 06 Aug 2015 11:29:23 +0200
Subject: Looking for Ada Coding Standard from GSFC
Newsgroups: comp.lang.ada

The Ada Programming Wiki book refers to a number of coding guidelines[1].

Most of the links have been dead, but I could find working links for both the mentioned ISO standard and the ESA standard.

The GSFC Ada coding standard (Stephen Leake, NASA Flight Software Branch — Ada Coding Standard) seems only to be available at [2]. Does anyone still have an official link to which I can point the Wiki? Is this document hosted anywhere else?

[1] https://en.wikibooks.org/w/index.php?title=Ada_Programming/Coding_standards&stable=0#Coding_guidelines

[2] https://web.archive.org/web/20100527142102/http://software.gsfc.nasa.gov/AssetsApproved/PA2.4.1.1.1.pdf

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Thu, 6 Aug 2015 07:44:24 -0700
Subject: Re: Looking for Ada Coding Standard from GSFC
Newsgroups: comp.lang.ada

> [...]

There is a similar standard at

https://gds.gsfc.nasa.gov/code_standards_ada.pdf

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Fri, 07 Aug 2015 08:57:45 -0500
Subject: Re: Looking for Ada Coding Standard from GSFC
Newsgroups: comp.lang.ada

>> https://gds.gsfc.nasa.gov/code_standards_ada.pdf

> Thank you very much. The latter seems basically an updated and extended version of the former,

Yes; that is the version the GDS team actively used (at least, while I was there).

> so I could point the Wiki book directly at this version.

I've retired from NASA, and the GDS team may not be around much longer, so this site may disappear as well.

I have the LaTeX source, if anyone is interested.

Gnoga Tutorials

From: Pascal Pignard <p.p11@orange.fr>
Date: Sun, 16 Aug 2015 09:06:15 +0200
Subject: Re: tabs and cards
Newsgroups: gmane.comp.lang.ada.gnoga

Though in French language, you have this tutorial with all views described and an example:

http://blady.pagesperso-orange.fr/telechargements/gnoga/gnoga_wf.pdf

http://blady.pagesperso-orange.fr/telechargements/gnoga/hello4.adb.pdf

Ada Distilled 2012

From: Richard Riehle <rriehle@itu.edu>
Date: Wed, 19 Aug 2015 14:21:27 -0700
Subject: Ada Distilled 2012
Newsgroups: comp.lang.ada

I talked with Ed yesterday. He is working on the update. He is also including some new, fully coded and working, examples of the new Ada features. Some odd them will be in the new Appendix to the book.

Ed has created some new examples to highlight the significant changes in the language.

It might be ready in a few months, depending on how the testing of the new examples proceeds. Yes, we have had a long standing policy of ensuring the coded examples compile and execute as intended.

Meanwhile, I am thinking about teaching an on-line, for credit, graduate level (MS) course in Ada under the umbrella of the school where I am currently an adjunct faculty, International Technological University (www.itu.edu) in San Jose, CA. ITU is a non-profit school focused on engineering, IT, software engineering, and computer science.

If there is sufficient interest from the Ada community, including contractors and others, I may be able to persuade the school to let me teach this class on-line across international borders. You may send me an email at rriehle@itu.edu, if you have some people who would want to enroll in such a course.

Ada Inside

Job: SPARK for Mobile Payment Services

From: Springboard Worldwide
Date: Wed May 13 2015
Subject: High Integrity Software Developer Ada - SPARK
URL: http://springboardww.com/index.php/recruitment/current-vacancies/high-integrity-software-developer-ada-spark

Our client delivers true innovation in the mobile payment / mCommerce space – levering advantages from its core IP to create unique-to-market products and services that deliver secure mobile payments direct to bank accounts, eliminating several barriers to growth in the sector. This is an unrivalled opportunity for a talented software developer with Ada/SPARK experience to join their development team based in Newcastle.

Main responsibilities:

- Work within expanding multi-skilled Agile delivery team to design, architect and develop innovative products
- Apply your take on Agile Delivery, adopting SCRUM techniques to develop great products
- Work to a Waterfall methodology when Agile is not appropriate
- Contribute to estimates, be involved in planning phase for sprints and to put self forward to take ownership of tasks, rather than await allocation of said tasks
- Manage own workload and be able to progress tasks, use initiative, be pro active and to report on progress of tasks in daily SCRUM
- Unit test own and peer development
- Understand full product portfolio rather than limit knowledge to products/modules you are working on

[...]

MAT - the Memory Analysis Tool

From: Stephane Carrez <Stephane.Carrez@gmail.com>
Date: Mon May 25 2015
Subject: Using MAT the Memory Analysis Tool
URL: http://blog.vacs.fr/vacs/blogs/post.html?post=2015/05/15/Using-MAT-the-Memory-Analysis-Tool

MAT is a memory analysis tool that monitors calls to malloc, realloc and free calls. It works with a small shared library libmat.so that is loaded into the program with the LD_PRELOAD dynamic linker feature (See the ld.so(8) man page). The library overrides the malloc, realloc and free function to monitor calls to these functions. It then writes or sends probe events which contain enough information for MAT to tell what, when, where and by whom the memory allocation was done.

MAT will assign a unique number to each event that is collected. The tool will reconcile the events to find those that are related based on the allocation address so that it becomes possible to find forward and backward who allocates or releases the memory. When started, the tool provides a set of interactive commands that you can enter with the readline editing capabilities.

[...]

Gnoga Demo: Connect Four

From: David Botton <david@botton.com>
Date: Sun, 14 Jun 2015 05:22:35 -0700
Subject: New Gnoga demo
Newsgroups: comp.lang.ada

Pascal Pignard (now also one of the maintainers of Gnoga) has added a new demo app, Connect Four.

Originally developed for GNAT_JVM by Barry Fagin and Martin Carlisle, US Air Force Academy. Pascal adapted it to GNOGA and left the original GNAT_JVM code in place as comments, for comparison.

http://gnoga.com:8083

Examples of Using Ada for Prototypes

From: Simon Wright <simon@pushface.org>
Date: Wed, 29 Jul 2015 09:55:07 +0100
Subject: Re: If not Ada, what else...
Newsgroups: comp.lang.ada

> At the moment I'm working on a prototype where the production version most likely will be written in assembly or some highly processor specific language by the customer. But Ada is very practical for writing algorithms in a readable form.

I designed a Mascot[1] kernel for a dual-processor F2420 machine[2] in Ada in about 1985. There was a small part (context switching) where "at this point, a miracle occurs". The implementation (in assembler) had one error on delivery and was in service (I believe) up to 2011.

This led to unfounded rumours that we had an Ada compiler for the machine!

Interestingly, Ravenscar has some commonality with Mascot's approach.

[1] http://async.org.uk/Hugo.Simpson/MASCOT-3.1-Manual-June-1987.pdf

[2] http://www.cbronline.com/news/ferranti_offers_f2420_at_five_times_power_of_fm1600e

Cubesat to the Moon

From: Peter C. Chapin <PChapin@vtc.vsc.edu>
Date: Wed, 19 Aug 2015 21:08:10 -0400
Subject: Re: If not Ada, what else...
Newsgroups: comp.lang.ada

> [...]

I am involved with this project at Vermont Technical College where I am working with a small group of students on the flight control software. We have a web page here:

http://www.cubesatlab.org/

and a blog here:

http://cubesatlab.blogspot.com/

Both are a bit sketchy at this time but we hope to enhance them in the coming months. The project is looking at a 2018 launch on SLS with spacecraft delivery in the second half of 2017. We are in the early stages of development... in fact we are still gathering basic requirements. It should be fun!

Ada in Context

Preconditions, Postconditions and Side Effects

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 6 May 2015 16:07:17 -0500
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

> Therefore, I don't think that it is the right choice for a language that is mainly used in safety critical systems.

If that's all Ada is going to be used for, it's completely irrelevant what features it has. I would never have used Ada personally if that was the case. Ada (IMHO) is a language to write (more) correct programs, no matter what kind of programs you write.

The basic idea behind the preconditions in Ada 2012 is to give a way for people (and implementations and tools as well) to ease into using additional checking and proof. Almost no one is willing to submit to the horrors of complete description of entities as required by SPARK. But (almost?) everyone using Ada would be interested in improving the correctness of their programs, one assertion at a time.

After all, consider how you likely learned the value of ranges and strong types and subtypes. Most everyone started out using mostly type Integer for everything. But one quickly notices that those cases where separate types were used get more errors detected at compile time and at runtime -- eliminating what usually are long and painful debugging sessions. This positive feedback loop quickly turns most Ada programmers into advocates and heavy users of the simple tools available in Ada.

By extending those mechanisms (via predicates and preconditions) to arbitrary expressions, we allow much more such error detection to occur.

There's also a performance benefit. In my experience, 10-20% of Ada code is checking code correctness (such is that a container routine is not passed a null cursor). By making these sort of rules preconditions or (better) predicates, we increase the chances that errors are detected immediately so no debugging is needed.

I don't believe that checks in Ada (of any kind) should ever be turned off. It's much better to let the compiler eliminate those that aren't needed (which is most of them, if your program is written correctly). The same will happen for predicates and preconditions and so on.

I also don't believe in separate proof tools. That's something that should be a basic part of the compiler (it has to be to do optimization, check elimination, and the like anyway). The difficult question is how to feed information about those things (particular checks known to fail) back to the programmer (as optimization phases tend to run without messages, and the messages that they do give are rather non-specific). In order for proof to be part of the compiler, the proof language has to be part of the language.

Lastly, fancy proof languages tend to be beyond the skill level of ordinary (and some not so ordinary) programmers. Despite, 6 years of University education, a masters degree, and 30 years of real-world experience, I had to have both the meaning of the implication operator and a "universally quantified predicate" explained to me. As it turns out, I had run into both in the past, but not under those names. Moreover, the programming language semantics is already complete -- why invent a new syntax just to confuse people? (I know from the early days when the compiler was written in Pascal how hard it is to switch between two similar languages used in similar contexts. That would be a permanent rather than transient problem.)

As I noted before, Ada (prior to 2012) uses exceptions to describe both requirements on callers and error conditions in a routine. It's much better to separate these, because the former can be eliminated by proof techniques and the latter cannot (no proof technique can tell you that a file will exist when it is initially opened). Ada 2012 preconditions allow one to do this without having to change the defined semantics of a routine (meaning that they can be profitably used on existing code).

The idea that proof has any value by itself is the real problem here. At best it is a tool to reduce the needed checking in a program, and a way to detect problems at compile-time (in this later use, it's only really of value as part of the compiler -- most programmers will not screw around with extra tools that have to be configured and managed and slow down the development process even more). Once you over-rely on proof, all you've done is forced your code into a new kind of specification, one that will have at least as many errors as the original. There's little value in that (especially in larger programs).

Anyway, more than enough ranting on this topic. IMHO, Ada 2012 gets it right, and building SPARK on top of it makes it more accessible to more programmers. That seems like a good idea, even if SPARK itself remains misguided.

From: Stefan Lucks <stefan.lucks@uni-weimar.de>
Date: Thu, 7 May 2015 12:06:29 +0200
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

[...]

> Almost no one is willing to submit to the horrors of complete description of entities as required by SPARK.

Actually, you don't need *complete* descriptions for SPARK. Often, verifying incomplete descriptions can be useful. Of course, the static verification will only verify the properties you describe.

[...]

> I don't believe that checks in Ada (of any kind) should ever be turned off.

Here, I heavily disagree. Often, checking relevant properties is much too expensive to perform the checks them in production code.

A simple example is binary search over a sorted array. The precondition requires the array to be sorted. If the compiler succeeds in optimising the test away, it is equivalent to a static program verifier proving the precondition holds when the binary search is called. If the compiler fails to optimise the check away, the execution time goes up from logarithmic to linear. If you can live with that, you don't need binary search!

Actually, one thing I am missing from Ada 2012 is a convenient and fine-grained way to tell the compiler which pre- and postconditions and invariants are to be checked, and which checks are to be skipped.

Most urgently, I would expect an option to skip checking ordinary pre- and postconditions, without skipping those that explicitly raise some exceptions. The point is, these two forms of precondition are semantically totally different:

[...]

Maybe, Ada 202X could include something like

```
with Pre => ... -- plain precondition, can
                -- be turned off

Pre'Check => ... -- must be checked at run
                 -- time
```

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 7 May 2015 14:16:54 +0200
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

[...]

> A simple example is binary search over a sorted array. The precondition requires the array to be sorted.

And what does it mean for the behavior? If unsorted input is *valid* and *must* raise exception (contracted behavior) then you cannot remove code *implementing* this behavior.

Consider this client program:

```
declare
   X : Element;
begin
   X := Search (Data, Key);
   begin
      null;
   exception
      when Not_Sorted_Error =>
         X := Search (Sort (Data), Key);
   end;
end;
```

Is this code correct?

[...]

From: Stefan Lucks <stefan.lucks@uni-weimar.de>
Date: Thu, 7 May 2015 20:00:54 +0200
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

[...]

> Consider this client program:

[...] I'll slightly rewrite your client program -- I believe, the exception handler was at the wrong place.

```
declare
   X : Extended_Index_Type;
begin
   X := Search (Data, Key);
exception
   when Not_Sorted_Error |
        Assertion_Error =>
      X := Search (Sort (Data), Key);
end;
```

> Is this code correct?

It depends on the specification of "Sort".

Specification 1:

```
function Search(Data: Array_Type;
                Key: Value_Type)
             return Extended_Index_Type
with
   pre  => Sorted(Data),
   post => (if Data(Search'Result) in
               Data'Range
            then Data(Search'Result)=Value
            else Search'Result = No_Index
            and then
               (for all I in Data'Range =>
                     Data(I) /= Key));
```

The expression "Sorted(Data)" is a precondition. Every client which can possibly violate the precondition is buggy. Thus, the above code is buggy. (Or otherwise, the exception handler is dead code.)

One property of a proper precondition (or postcondition or ...) is that disabling the check does not change the functional behaviour of correct programs.

Specification 2:

```
function Search(Data: Array_Type;
                Key: Value_Type)
             return Extended_Index_Type
with
   pre  => (Sorted(Data) or else
            raise Not_Sorted_Error),
   post => (if Data(Search'Result) in
               A'Range
            then Data(Search'Result)=Value
            else Search'Result = No_Index
            and then
               (for all I in Data'Range =>
                     Data(I) /= Key));
```

The expression following "pre" is "contracted behaviour" as you put it. The code above is correct, and disabling the check must be prohibited, because it would break correct programs. Which is why I wrote the following:

>> [...]

In other words, I want to be able to switch of proper preconditions (and postconditions, whatever) without affecting contracted behaviour.

I depreciate the usage of the word "precondition" for the expression following "pre" in spec 2. But I will not fight about names.

Furthermore, I am quite happy with Ada allowing to specify contracted behaviour, even I would have preferred use an aspect of its own for contracted behaviour. The "pre" aspect should better be have been reserved for proper preconditions. But this appears too late now. On the other hand, it would not be too late to support disabling proper preconditions without changing contracted behaviour.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 7 May 2015 14:01:34 -0500
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

> One property of a proper precondition (or postcondition or ...) is that disabling the check does not change the functional behaviour of correct programs.

Sure, but this is irrelevant. There are no "correct" programs (in all possible senses). How do you know that you have a correct program? If you use some prover program, it too may have bugs. Or there might be bad data (cosmic rays?) Or the specification might be incomplete. Consider the latest Dreamliner issue; that probably wouldn't have been caught by a prover simply because no one would have included an appropriate assertion.

Ergo, I don't believe that "proper preconditions" really exist. And in the rare cases that they do (perhaps because of an immediately preceding postcondition), a compiler would have eliminated them anyway, so you're not paying anything for the supposed runtime check. (After all, Ada compilers have been aggressively removing checks since 1983; that's nothing new to an Ada compiler writer.)

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Thu, 07 May 2015 22:29:00 +0300
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

> [...] There are no "correct" programs (in all possible senses). [...]

Those are practical problems, not problems of principle. If you would take the same attitude to mathematics, you would claim that there are no correct theorems. So I disagree with you.

> [...] Consider the latest Dreamliner issue; [...]

If it was an overflow problem (and not wrap-around of a modular type) CodePeer would probably have complained that it could not prove absence of overflow. That is, many failures result from violations of general assertions that one does not have to write explicitly.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 8 May 2015 18:16:27 -0500
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

> [...]

All programming is practical. We do not care about theorems, only that the current program is correct in the current environment. Everything has to be reproved when anything changes (another good reason for putting it into the compiler, as skipping the step isn't possible, and thus problems like the Ariene 5 don't happen).

It seems that most of you here are infected with the "theory" disease. I want to make practical programming better, and I don't give a damn about any stupid theories. Maybe I'm just getting crazy in my late middle age. :-)

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Sat, 09 May 2015 08:18:37 +0300
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

> [...]

It is true that an optimizing and run-time-check-removing compiler has to make the same kind of analyses and have the same kind of understanding of the program's semantics as a program-proving tool. But constructing a proof from scratch is expensive, unpredictably expensive, and possibly non-terminating, while compilation of source code into machine code should take a time that is reasonable, and reasonably predictable.

Hitherto, compilers have been expected to remove run-time checks that the compiler can prove to itself are redundant, but not to remove *all* redundant run-time checks. If the latter is required, compilation time becomes unlimited -- a practical problem, no?

To make proof a routine part of compilation, it has IMO to be reduced to *proof checking*. Checking a proof is fast and terminating.

To integrate proof-checking with compilation, the programming language has to be able to express the proof (axioms, lemmas, individual proof steps) interwined with the expression of the computation that is to be proved. And this has to be so easy that it tempts the programmer to write the proof -- or at least enough of the proof to guide the compiler -- as a routine part of creating the program. (Echoes here of proof-carrying code, http://en.wikipedia.org/wiki/Proof-carrying_code.)

Ada has always had such features -- principally types, subtypes, ranges -- and Ada 2012 has added more -- pre/post-conds and invariants. However, I'm not sure if the features are yet sufficient to let us require, in the Ada standard, that an Ada compiler should be able to prove (rather, to check) exception-freeness, or termination, just to give two examples.

I believe it is a good goal for evolving Ada, but of course not the only goal.

By the way, if exception contracts are added to Ada, termination contracts should be considered, too.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 11 May 2015 19:15:27 -0500
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

[...]

> [...] constructing a proof from scratch is expensive, unpredictably expensive, and possibly non-terminating, while compilation of source code into machine code should take a time that is reasonable, and reasonably predictable.

I don't see that either is a given. Janus/Ada would run approximately forever if we didn't artificially bound the optimization time, and it still can take a long time to produce code. If we ever built the link-time code generation version, that time would go up by a lot.

As with everything, one can make bad code quickly, or take longer to make good code.

Similarly, I don't see any reason that proper proofs should take forever, as it is approximately the same problem as optimization and code generation. At some point, you give up and decide that something is unprovable. No big deal.

> [...]

If you want truly good code, compilation time should be nearly unlimited. But I agree that there is a practical limit, but the same limit applies to a proof tool (if you can wait 12 hours for a proof tool, you can wait 12 hours for a compilation, too, especially if one can turn that mode off or down, just like optimizers).

> [...] I'm not sure if the features are yet sufficient to let us require, in the Ada standard, that an Ada compiler should be able to prove (rather, to check) exception-freeness, or termination, just to give two examples.

Certainly not yet. One needs exception contracts at a minimum, as otherwise one cannot tell between exceptions raised as part of the behavior of a subprogram and those which represent bugs.

> [...]

I doubt that there is an "only goal", because lots of people have input. I happen to think it is the only goal that ultimately matters, as much of the other ideas don't really move the needle in any significant way.

> [...]

[...] I don't quite see the point of termination contracts, a non-terminating subprogram is wrong 99.9% of the time. Maybe a "non-termination" contract would make some sense?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 7 May 2015 13:52:25 -0500
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

> [...] If the compiler succeeds in optimising the test away, it is equivalent to a static program verifier proving the precondition holds when the binary search is called.

Exactly! That's the entire idea; the compiler *should* be doing these optimizations, indeed one major branch of static program verification comes from enhancing compiler optimizer technology (CodePeer is an example of that). I think that technology should simply have stayed in the compiler.

> [...]

If the compiler fails to optimize the check away, your program is wrong in some sense, and you should have gotten an error or warning (depending on the compiler mode and exception contracts) to that effect. You ought to fix your program (probably by adding an appropriate predicate) so that the check *can* be eliminated (or pushed to somewhere where the cost is irrelevant).

[...]

Anyway, the Assertion_Policy can be changed locally, and the policy in effect at the point of the declaration determines what policy is used for calls. Plus the policy can be set separately for different kinds of assertions. Thus, you can get the effect you want with the existing policies, so long as you don't try to write two different kinds of assertions on the same subprogram.

Note that there is some debate about the value of the fine-grained policy setting,

it's unclear that GNAT implements it correctly. If some of their customers showed concern about the correct implementation of those rules, that certainly would change.

From: Stefan Lucks <stefan.lucks@uni-weimar.de>
Date: Thu, 7 May 2015 21:40:27 +0200
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

> If the compiler fails to optimize the check away, your program is wrong in some sense, [...]

I am a big fan of correctness proofs, where they are applicable. But logically,

```
Not(Proven_Correct) /=
        Proven(Incorrect)
```

Furthermore, automatic theorem proving can only go so far. I may actually know my program to be correct -- and maybe I can even prove it manually. Why should the compiler reject my program, or insert useless checks, just because it fails to find the proof?

Warning or not I would consider a compiler (or a language) which generates linear-time code for binary search badly broken. Rejecting the program would be the lesser evil. Which would turn Ada into a new SPARK.

But then, the Ada standard would have to define the underlying theorem prover, for compatibility reasons. Else, the same program may be proven correct by one prover, where another prover fails.

> You ought to fix your program [...]

Why do I need to fix the program, if I know it is correct? Just because the compiler isn't good enough at theorem proving?

[...]

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 8 May 2015 17:58:54 -0500
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

> [...]

It's certainly true that there is a potential portability problem here, but I think we have no choice but to allow it. Otherwise, we have to force these sorts of things into warnings, which means that they have no real force and worse that the Standard cannot talk about them. In such a case, programming will never get better.

> [...]

No, the problem is in portability. The standard cannot get involved in what can and cannot be proven (other, perhaps than setting some minimum requirements). Beyond that, through, it has to be implementation-defined. So the question boils down to do we allow one compiler to reject a program because it can't be proved that it does not raise an exception (for example) while another compiler allows it because it can prove that? I think we HAVE to allow that sort of non-portability; for one thing, it gives vendors a serious incentive to improve their compilers. On top of which, it is idiocy to require a compiler to reject something that it can tell is not a problem. (Particularly, something optional like exception contracts.)

I expect this to be the defining question of Ada 202x; we can't have exception contracts without deciding this question somehow, and I don't see much possible advancement for Ada without those contracts.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 11 May 2015 19:28:32 -0500
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

> [...] Don't you write tests?

Outside of the ACATS (which is a product unto itself), not often. Sometimes I rewrite other people's tests to generalize them or to simplify them.

Most of my programs are tested in-situ, either with live data (as in the web server/mail server) or by using constructed data. (The ACATS essentially falls into this category, from the perspective of a compiler.)

I definitely don't write or use unit tests in the majority of cases. It's easier to use live data than to figure out some way of getting that data correctly initialized in order to do a unit test. (Consider operations on a compiler symbol table. In order to unit test those, you have to construct a symbol table for them to work on, and that symbol table has to be constructed exactly as the compiler will do it - otherwise you end up testing the unit test more than anything useful. Since the compiler already knows how to do that, we let the compiler do the setup and then debug in place.)

Since testing proves almost nothing about a program's quality, I prefer to avoid them (and it) as much as possible. [Probably too much. :-)] I want my compiler to detect all of my mistakes before I run anything. That's the whole reason I started to use Ada and continue to use Ada. (Plus testing and debugging of tests is incredibly frustrating.)

> Most interesting programs cannot be debugged. It is quite strange, you don't believe correct program exist, but trust "debugged" programs do! (:-))

Now, Ada does my debugging. When there's a problem that actually requires debugging, it takes forever for it to get fixed. (Sorry, Tero. ;-) My point as just that a fielded program already has some amount of testing and fixing done to it (we hope), and once that happens, leave it alone until/unless there is a problem.

From: Georg Bauhaus <bauhaus@futureapps.de>
Date: Thu, 07 May 2015 23:29:03 +0200
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

> If the compiler fails to optimize the check away, your program is wrong in some sense, [...]

With respect, this attitude towards manipulating a program's meaning during translation makes a compiler assume responsibility where it really is the programmers' responsibilities (as per contract), as expressed in clauses like Pre and Post.

Why then is the *program* somehow buggy, as you say, because some compiler's optimizers can't follow the math that has been done already, and expressed as a truth in Pre?

As Stefan Lucks explains in his reply, why would a compiler override what the programmer has stated as a proven truth? That's not Ada. That's more like a compiler getting in the way.

I think that

```
Pre'__unchecked__ =>
```

might be in order, then, with the understanding that it's not real, but conveying the idea.

Thus, if the compiler puts checks where the programmer has shown they are superfluous, that's not Ada. At least it used not to be like that.

"Design your program by obeying our optimizer, be defensive, don't bother with logic and proofs! We'll take care. Doing so prevents bugs (if possible)."

That's not Ada. That's another sales strategy.

Are we supposed to forget such contracts entirely because this kind of formally proven programming is, as you say, not meant by Ada's new "contracts"?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 8 May 2015 18:11:16 -0500
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

> [...]

There is no "truth" in Pre; it's just part of the description of the meaning of the program. It's madness to assume anything more, you WILL be burned.

And there is little value (IMHO) to separate proofs. At best, you're proving what the separate tool *thinks* the semantics should be. Whereas the compiler actually *knows* what the semantics are.

Separate tools like SPARK have value today because compilers (and most languages!) aren't smart enough to be able to apply proving technology to the generation of programs. (Mostly because of performance concerns.) But that should change over time, and there shouldn't be any reason to keep them separate.

It's possible of course that I've reached my expiration date in terms of where Ada (and programming languages in general) need to go. Especially as most code is mashed-up today and as such is barely functional -- correctness is irrelevant when it barely meets any need. But then I (and most of us, in fact) have no future (and I worry about the future of humanity as a whole).

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 8 May 2015 17:52:15 -0500
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

> [...] Why do I need to fix the program, if I know it is correct? Just because the compiler isn't good enough at theorem proving?

Because there is no such thing. There is no real way for you to know that it is correct. I have plenty of examples of supposedly correct programs that turned out to have serious bugs.

The only way for a program to be "correct" is for it to be proven that way by the compiler. (And even then, the compiler algorithm might have problems.) Because if any other tool does it, the compiler might disagree (because either tool has bugs), and thus the actual code might still in fact be incorrect.

>*IF* there is a problem at all.

See above. There is *always* a problem; the only question is whether you are willing to defend against it or not.

For example, in this "Is_Sorted" example, if you are assuming that some object is sorted, then it should have a predicate to that effect. In such a case, the compiler would be able to eliminate all of the precondition checks on calls, the only time the predicate would be checked is when the object is initially created (which certainly should not be on any critical path), or if it is modified by a routine that doesn't include Is_Sorted in its postcondition (which is clearly the source of potential bugs as well).

In the absence of those sorts of things, you are just hiding potential flaws.

From: Stefan Lucks <stefan.lucks@uni-weimar.de>
Date: Mon, 11 May 2015 12:35:47 +0200
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

> [...] There is *always* a problem; the only question is whether you are willing to defend against it or not.

Yes, and no. Having worked a bit with SPARK, my experience is mixed. When the tool fails to verify my program, the investigation sometimes actually reveal a subtle bug, and I am happy to have used the tool. But quite as often, the verifier just fails at some simple and indisputable facts, such as proving that ((p-1) + 1) mod p is zero. [...]

So the ability of any of the tools we have now (and, I would argue, in the foreseeable future) to prove program correctness is very limited. If a compiler with such limited abilities turns my logarithmic-time search routine into a linear-time routine, just because it couldn't prove that the input to the routine is always sorted, then the compiler is broken.

Proving program properties (apparently you don't like the word "correctness" in that context) is a *tool* for the programmer. If properly used, it can be an extremely useful tool, especially for medium- and high-assurance software. But doing foolish things if the proof fails, or strictly requiring all relevant properties must actually be proven would turn this tool from something useful into a terrible burden.

[...]

[Primes:] In practice, people usually call the Miller_Rabin primality test:

```
function MR_Is_Prime (N : Num;
                      Repetitions : Natural := 500)
return Boolean
   with Post => Miller_Rabin_Prime'Result =
         Is_Prime (N);
```

As a specification, the postcondition is useful. For testing with tiny numbers, it might be OK. But for testing with realistically-sized N, or for production code, this test *must* be deactivated. The user cannot wait for Is_Prime to terminate.

By your logic, disabling the test would be bad. Thus, the compiler would eventually have to prove the fact that the Miller-Rabin test is mathematically correct, and always gives the proper result, right?

But proving such properties is mathematically deep, and way beyond the abilities of the theorem provers we actually have. (And I would not hold my breath to wait for this to change.)

Even worse, the Miller-Rabin test is a probabilistic test. There is some chance that a composed number actually passes the test. The chance is very small -- for the default Repetitions, the chance is below 2^{-1000}, so neglecting this risk is perfectly OK. But usual tools for program correctness (or for program properties) are not able to deal with probabilistic results.

> In the absence of those sorts of things, you are just hiding potential flaws.

Agreed! But at some point, someone (some human, not the compiler!) has to make the choice how to proceed, if a relevant property has not been formally proven. Neither simply rejecting the program, nor inserting a potentially huge ballast of additional test, is always an option.

The language, the compiler, and other tools, may support you to write good software, especially medium- and high-assurance software. Ada has been a great tool for that purpose from its very beginning, and Ada 2012 has made a significant step forward in that direction. But expecting too much from the tools them will very quickly get into your way, and eventually also destroy the support for the tools.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 11 May 2015 20:03:52 -0500
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

> [...]

That's the value of those tools: to prove that something is not correct. It's just like testing in that way; you can't really prove that the program is correct, but you surely can prove that it is not correct.

> But quite as often, the verifier just fails at some simple and indisputable facts, [...]

These sorts of problems would come up in proving postconditions, but I don't see that happening for preconditions.

> If a compiler [...] turns my logarithmic-time search routine into a linear-time routine, [...] then the compiler is broken.

No, I'd still argue your code is broken. If *you* know that some object is always sorted, then *you* should tell the compiler that with an appropriate predicate:

```
subtype Sorted_Array is Some_Array
   with Dynamic_Predicate => Is_Sorted
         (Sorted_Array);
My_Array : Sorted_Array := ...;
```

Now, whenever My_Array is assigned (as a whole) or passed as a parameter, it will be checked for whether it is sorted. That pushes the check to whenever the array is created/initialized/modified, which is not going to have any effect on your sort routine.

On top of which, the routines that do the creation/initialization/modification probably ought to have postconditions that the array is sorted as well. In which case, the object also will have been previously checked, so the cost will be at the end of those routines. And possibly (although unlikely in the particular case),

that check could be proved away there as well.

> [...] But doing foolish things if the proof fails, or strictly requiring all relevant properties must actually be proven would turn this tool from something useful into a terrible burden.

No real burden, IMHO. The sorts of properties that should be involved should be relatively simple to express and thus prove, and not that expensive to check. Much like null pointer checks or variant checks in Ada. Turning these sorts of things off is silly.

I can see that are some cases where the properties are too expensive to verify at runtime. It would be nice if there was a way to turn off those (AND ONLY THOSE) properties. But Ada doesn't have that sort of granularity, so I wouldn't bother writing them in the first place. (At least not as preconditions; most of my programs have extensive tracing/self-checking modes that can be enabled unit-by-unit; that's the place for such expensive things.)

[Primes:] If it hurts, don't write that. :-) I don't begin to believe that all program properties can be proved. Indeed, there is a major problem in that there is no good way to specify which properties that a subprogram does *not* affect. There is an infinite number of such properties, so specifying them one by one in the postcondition:

```
Is_Sorted (Arr) = Is_Sorted (Arr)'Old and ...
```

is madness. (And even in a small system, there are a lot of properties. Consider just the interesting properties of a Vector container. The length, capacity, and tampering state all immediately come to mind, and most routines change none of them. How to communicate that?)

[...] I think any useful postcondition has to be reasonably executable. Else there is no difference from a comment, and you would be better served using a comment as won't throw out all of the easy checks with this silly one.

[...] *writing* a test that you can't actually execute is bad, as it tells no one anything useful.

[...]

If you can't execute it, and you can't prove it, what exact good is it over having a comment

```
-- Returns True if N is prime.
```

???

I can't think of any.

[...]

I don't think anyone would ever want a system that *required* proving everything. The important thing IMHO is that you can find out what wasn't proven so you can determine whether to fix it (via additional specifications) or whether to ignore it (the obvious case being that the unproven case is on a non-executable path).

> [...] But expecting too much from the tools them will very quickly get into your way, and eventually also destroy the support for the tools.

True enough. Expecting proof to be anything more than another way to determine errors early is the root of the problem. It's useful to know what the compiler doesn't prove in order that one can decide to ignore it, but clearly ignoring it is the default (just as it is for regular Ada runtime checks). No one should ever be forced to change any code unless a proof determines that a check *will* fail (or there is a possibility of raising an uncontracted exception -- but no one would ever be required to use an exception contract).

I want to bring this tool to people that would not go out of their way to use it. (I.e., me. :-) That means it has to be in the compiler so that they get the benefits automatically, because there is no way I'm going out and buying (plus learning) a separate tool to do those things. I suspect that many (perhaps most) programmers are like me. Like everyone, I want it all, for free, right now. :-) Only Ada comes close, and I just want to make it closer.

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Tue, 19 May 2015 09:46:07 +0200
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

> (for all I in 2 .. N-1 => (N mod I) /= 0)
>

> You don't need a profiler to figure out that this is prohibitively slow for largish N, do you?

I don't need a profiler to estimate that it takes long time to execute, but I need a profiler to see where the compiler can't eliminate it from a critical path through static analysis.

>> I am aware that we currently don't have as fine-grained control of assertions as that would require to work well, but I assume that this is something that can be discussed with the ARG and the compiler vendors.

>

> This is precisely my point!

Good. I noticed an interesting proposal for an extension to the assertion policy control in one of the posts in this thread. I suppose we should push to have the ARG accept this (or something similar).

From: Bob Duff <bobduff@theworld.com>
Date: Tue, 12 May 2015 10:21:41 -0400
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

> [...] It would be nice if there was a way to turn off those (AND ONLY THOSE) properties. But Ada doesn't have that sort of granularity,

Sure it does. If Is_Sorted is too slow for production use, you can say:

```
... with Predicate => (if Slow_Mode then
        Is_Sorted(...))
```

and set the Slow_Mode flag to True for testing. Also set it to True when running proof tools.

Alternatively, you can say something like:

```
function Sort (X : My_Array) return
        My_Array
  with Post => (if X'Length <= 20 then
        Is_Sorted (Sort'Result));
```

Now calls to Sort can be O(log N) instead of O(N). And if Sort doesn't do anything special for arrays longer than 20, the postcondition is likely to catch any bugs in Sort.

> [...] I don't begin to believe that all program properties can be proved.

Yes, that's obviously true. Here's a property of GNAT:

Compared to most compilers (for any language), GNAT usually gives better error messages.

Anybody who has used GNAT and a lot of other compilers knows that property is true. But nobody can prove it in a mathematical sense, because there's no way to formalize it.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 12 May 2015 17:37:49 -0500
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

> [...]

Of course. That's essentially what I've ("we've", really, Isaac created a lot of the tracing stuff in Janus/Ada) been doing for years. I just hadn't thought of trying to use it directly in the assertions. We'd use a function call, though, rather than a constant:

```
... with Dynamic_Predicate =>
        (if JTrace.Trace (Current_Unit)
        then Is_Sorted (...))
```

When compiled for testing, JTrace.Trace is a function call which returns true or false based on the selections from a tracing menu that pops up when the tracing options are used. When compiled for production use, Trace is an array with all of the elements set to False. (At least that was the idea, I don't think we ever used it that way.)

The downside here is a bit more noise, but the upsides are obvious (Stefan explained them in gory detail). One probably could design something shorter with the same effect (that would cut the noise).

From: Georg Bauhaus <bauhaus@futureapps.de>
Date: Wed, 13 May 2015 08:58:19 +0200
Subject: Re: {Pre,Post}conditions and side effects
Newsgroups: comp.lang.ada

> [...]

>

> ... with Dynamic_Predicate => (if JTrace.Trace (Current_Unit) then Is_Sorted (...))

Given this fine-grained run-time configuration (another IF and then a little something like a debugging thing from an implementation), is the condition in the same category of expressions as Is_Sorted?

The second, Is_Sorted, is strictly about the parameters, contractual, so to speak. The first looks rather different and distracting to me.

But in any case, then, maybe having a way of influencing the selection of checks could be expressed as

```
pragma Assertion_Policy (Post => Check
       and not MR_Is_Prime'Post);
```

Stipulating that policy_identifier in Assertion_Policy becomes just a little more flexible by turning the conditional into a portable feature specifiable outside the contracts, but near them:

```
pragma Assertion_Policy (
      assertion_aspect_mark =>
        policy_setting
    {, assertion_aspect_mark =>
        policy_setting});

policy_setting ::= policy_identifier { and
        mute_list }
          mute_list ::= not
        defining_identifier'matching_mark
       {, not
        defining_identifier'matching_mark
```

An Element of a Coding Standard

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 7 May 2015 14:09:31 -0500
Subject: Re: Polymorphism
Newsgroups: comp.lang.ada

[Public or private overriding operations.]

> Moreover, it probably would make sense to move all declarations of overriding into private

This is my coding standard. I only put new stuff into the visible part.

[...]

> because the fact of overriding is mere an implementation detail, since the primitive subroutine is there anyway no matter what.

Correct; that's why I put them into the private part.

Time to Start Over?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 28 May 2015 17:46:26 -0500
Subject: Re: Build language with weak typing, then add scaffolding later to strengthen it?
Newsgroups: comp.lang.ada

> [...]

Uh-huh. Janus/Ada 83 fit on and ran on floppies. (Heck, there wasn't anything else available on early MS-DOS.) Even the relatively tiny 5 1/4" floppies. It still would if you could find a machine that has floppies. The bloat is in GCC, not necessarily in Ada.

The bloat in the Standard (such as it is) came from adding lots of stuff that people thought was necessary (but arguably isn't): tagged types and dispatching, interfaces, prefix calls, assertions, and (especially) containers.

> So please explain to me why on earth I or DoD or ISO or anybody else needs all the bloat to confuse us.

We don't. As with all old languages, it's political. We can't remove old features (as that would break existing programs), so the only possibility is for the Standard to get bigger. It's also getting bigger because we've (me in particular) have been insisting on adding wording to fill holes, rather than just ignoring them. Based on my experience, I think a language standard with 17 pages is about 90% hole (unless, of course, the language doesn't actually do anything).

I'd think it's getting close to time to start over with Ada, not because of any major problem, but simply the accumulation of cruft. The problem is that if you think it's hard to convince people to use Ada with all of its track record, try doing that with a new language with no record. So I don't think there would be much of a market for that.

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Fri, 29 May 2015 11:31:31 +0200
Subject: Re: Build language with weak typing, then add scaffolding later to strengthen it?
Newsgroups: comp.lang.ada

> [...]

But aren't the existing programs being compiled with compilers for the appropriate (old) versions of the language?

How large is the actual benefit of maintaining practically full backwards compatibility?

Isn't it more a matter of not being able to agree on what is important to keep, and what isn't?

> [...] accumulation of cruft. [...]

Isn't that in itself an argument for letting Ada 2020 be a major change, where backwards compatibility isn't as important as using our current knowledge to improve the language? I wouldn't want an Ada 2012 program to be accepted by an Ada 2020 compiler with a different meaning, but I wouldn't mind it if the Ada 2020 compiler told me that I have to do things differently in Ada 2020.

From: Björn Lundin <b.f.lundin@gmail.com>
Date: Fri, 29 May 2015 12:56:06 +0200
Subject: Re: Build language with weak typing, then add scaffolding later to strengthen it?
Newsgroups: comp.lang.ada

> But aren't the existing programs being compiled with compilers for the appropriate (old) versions of the language?

Not necessarily.

Our system often has a life expectancy of 15 +- 5 years.

But the os/database/other tools does not.

Once say Oracle say - no support available - which they do relatively fast - most customers wants a platform upgrade.

That is - make the same system run on a newer os/db/whatever.

And that includes a new Ada compiler.

Ans one really nice thing about Ada is that is usually compiles and works right away.

If compiler-vendor-change took place, some fiddling is usually present, but upgrading a system from one GNAT version to another is painless.

You get tons of more warnings - and that is it.

Another scenario is when a customer wants some 'newer' technology, like webbish stuff.

To add AWS a relatively new compiler is needed.

And that should compile that rest of the system too - even if old.

> How large is the actual benefit of maintaining practically full backwards compatibility?

To us - very large.

> [...]

If you ask around enough, you will likely get answers that wants to keep 'odd' features. I - for example - love the separate construct. we use it a lot. But I think not too many use it.

From: Peter C. Chapin <PChapin@vtc.vsc.edu>
Date: Fri, 29 May 2015 08:03:08 -0400
Subject: Re: Build language with weak typing, then add scaffolding later to strengthen it?
Newsgroups: comp.lang.ada

> [...]

I also like using subunits now and then. It seems like a good fit when I have a package with one unusually large subprogram that dominates the package's physical content, but is just not important enough, or logically distinct enough, to make into its own unit.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 29 May 2015 16:45:58 -0500
Subject: Re: Build language with weak typing, then add scaffolding later to strengthen it?
Newsgroups: comp.lang.ada

> But aren't the existing programs being compiled with compilers for the appropriate (old) versions of the language?

Until they need some part upgraded. For example, I'm (slowly) moving my web and mail servers from Janus/Ada on a W2K machine to GNAT on a new Linux machine. It's likely that I'll want/need to use some new capabilities when that's done. Changing lots of code because someone didn't like some existing feature is unappealing.

Similarly, I know that pretty much any Ada code that I have will still work if I need it in some program. Reuse is a valuable benefit of Ada, and that cuts across time as well as projects.

> How large is the actual benefit of maintaining practically full backwards compatibility?

It's hard to say. Some people (Robert Dewar in particular) think we abandon compatibility far too easily in the ARG as it is. And we don't do that lightly as it is. He claims that compatibility issues discouraged many from using Ada 2005; the aspects and preconditions in Ada 2012 were enough to break through that barrier, but it would be easy for Ada 202y to fall into a similar trap (not enough important to make up for incompatibilities).

> Isn't it more a matter of not being able to agree on what is important to keep, and what isn't?

No. There definitely is a group that think that almost any incompatibility is unacceptable. And someone is using every core feature (even stuff like generic formal in out parameters); how do you decide who's code is not important enough to support.

> [...]

Sure, it's an argument. But what's typically happened when languages made big breaks is that the new version is much less used than the original. That goes all the way back to Algol 60 vs. Algol 68. I doubt Ada could survive a much less used version.

From: Peter C. Chapin <PChapin@vtc.vsc.edu>
Date: Fri, 29 May 2015 19:12:09 -0400
Subject: Re: Build language with weak typing, then add scaffolding later to strengthen it?
Newsgroups: comp.lang.ada

> [...] I doubt Ada could survive a much less used version.

I'd hold up Python as an example of how things don't work well when you make too many breaking changes. Python 3 is incompatible with Python 2, yet after 6.5 years there are still (many? most?) projects out there that require Python 2. I don't follow the Python community that closely but my impression is that the plan to entice everyone over to Python 3 failed. Now they are stuck with maintaining two incompatible versions of the language into the arbitrary future.

[The answer is “No”, it appears. —sparre]

Evaluation Order and Functions with “out” Parameters

From: David Botton <david@botton.com>
Date: Fri, 12 Jun 2015 08:56:23 -0700
Subject: Is this a bug in my code or the compiler?
Newsgroups: comp.lang.ada

Given:

```
function Token_Start (Source : in out
   Awesome.Source.Source_Type'Class)
   return Character;
function Token_End   (Source : in out
   Awesome.Source.Source_Type'Class)
   return String;
```

The following works:

```
function Get_Token_Text (Source : in out
      Awesome.Source.Source_Type'Class)
return String is
   N : Character := Token_Start (Source);
begin
   return N & Token_End (Source);
end Get_Token_Text;
```

The following does not work:

```
function Get_Token_Text (Source : in out
      Awesome.Source.Source_Type'Class)
return String is
begin
   return Token_Start (Source) &
          Token_End (Source);
end Get_Token_Text;
```

Token_End is never called and only the value of Token_Start.

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Fri, 12 Jun 2015 09:48:56 -0700
Subject: Re: Is this a bug in my code or the compiler?
Newsgroups: comp.lang.ada

> [...]

Given that the Source parameters to both Token_Start and Token_End are mode “in out”, I presume that both functions modify Source and therefore, Token_Start must be called before Token_End. Your 1st version ensures this order; your 2nd does not. You are presuming an order of evaluation of the parameters to "&" that is not guaranteed.

What happens if Token_End is called 1st?

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Sat, 13 Jun 2015 18:43:18 +0200
Subject: Re: Is this a bug in my code or the compiler?
Newsgroups: comp.lang.ada

>> Maybe there is an AdaControl rule to detect this kind of problem.

> Of course. It's called Parameter_Aliasing :-)

I tried it without luck on this test case:

% cat bad_style_2.adb

```
with Ada.Integer_Text_IO,
   Ada.Text_IO;

procedure Bad_Style_2 is
  function F (I : in out Integer) return
          Character;
  function G (I : in out Integer) return
          String;

  function F (I : in out Integer) return
          Character is
  begin
    return R : Character do
      if I < 0 then
        R := '-';
      else
        R := '+';
      end if;
      I := 2 * I;
    end return;
  end F;

  function G (I : in out Integer)
          return String is
  begin
    return R : constant String :=
          Integer'Image (I) do
      I := I - 1;
    end return;
  end G;

  C : Integer := 3;
begin
  Ada.Text_IO.Put_Line (F (C) & G (C));
  Ada.Integer_Text_IO.Put (C);
end Bad_Style_2;
```

% adactl -l 'check parameter_aliasing' bad_style_2.adb

%

I.e. no detection of “expression parameter aliasing” (or what we should call it). I think it is only slightly harder to detect than plain parameter aliasing, but I'm not yet quite proficient enough in ASIS to promise to contribute a new rule to AdaControl.

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Sat, 13 Jun 2015 22:50:05 +0200
Subject: Re: Is this a bug in my code or the compiler?
Newsgroups: comp.lang.ada

[...]

Yes, it works on procedure and entry calls, I didn't put (yet) the case of function calls... Added to my todo list for the next version.

From: Brad Moore <brad.moore@shaw.ca>
Date: Sat, 13 Jun 2015 13:42:10 -0600
Subject: Re: Is this a bug in my code or the compiler?
Newsgroups: comp.lang.ada

>> [...]

> Parallel evaluation of arguments.

In the above case with regard to implicit parallelism, the compiler should be able to determine that the two calls both involve modifications the same storage, which would be a data race, and then rule out parallelism and thus generate sequential code. So the issue here is not about parallelism, but about ordering of evaluation for the sequential case.

For the sequential case, I would think that a good compiler could also detect that an expression with multiple calls with in out parameters to the same storage is likely a problem with evaluation order, and generate a warning to the programmer, which could be averted by coding with "and then" for force evaluation order. If your compiler does not generate such a warning, it might be good to ask your vendor to provide such a warning.

Or a programmer could adopt a programming style to use "and then" for the general case, which I believe could be checked by a tool such as AdaControl.

This should work for the case Boolean sub expressions, but doesn't help in the case of concatenation operations. Hopefully, the compiler could at least generate warnings for this case, then the programmer can decide how best to address the warning.

[...]

>> Moreover, if B and A become pointing to the same object,

>>

>> Foo (A) and Foo (B) -- Legal, same effects

>>

>> Can a compiler detect this?

In Ada 2012, we have the attributes 'Has_Same_Storage and 'Overlaps_Storage.

These were introduced to facilitate writing preconditions for a subprogram. One would think that if these are available for checking multiple parameters of a subprogram, the compiler could also do similar checks for the parameters of subprograms that are part of the same expression. In some cases, this could be a compile time check, but in others, it may need to be a run-time check, that possibly could be enabled/disabled via compiler options.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Sat, 13 Jun 2015 22:21:46 -0500
Subject: Re: Is this a bug in my code or the compiler?
Newsgroups: comp.lang.ada

> A programmer having to worry about order of operation in a concatenation operation is a language flaw in my opinion. I can accept the issue in evaluation order of Boolean expressions or numerics (and knew of that), but not in non numeric types.

Lots of people would agree with you, but neither J. Ichbiah or STT would, so Ada doesn't define the order of evaluation of almost anything: parameters (not just concat, but in all subprogram calls), aggregate components, and many more things. It would be way too disruptive to try to introduce any such rules now, especially as there are legal dispatching calls that cannot be evaluated left-to-right.

The rules in 6.4.1 are supposed to prevent the worst problems, but of course they only work if correctly implemented.

I'd suggest submitting one or more ACATS tests. :-)

Contributing Ada Sources to a C++ and CMake Project

From: Alejandro R. Mosteo <alejandro@mosteo.com>
Date: Thu, 2 Jul 2015 03:44:12 -0700
Subject: Re: C++/cmake project, injecting Ada...
Newsgroups: comp.lang.ada

[...]

To summarize, the challenge was to contribute Ada code to a C/C++ project using the CMake build environment and with minimal disruption for all parts involved.

What I've found is that CMake 2.8 adds an "external project" build command that enables calling gprbuild with ease. If you're in a debian-based distro which packages gnat, the other contributors that are interested in compiling your Ada part just need to install a package. To me that qualifies as minimal disruption :)

Advantages to each side:

1) The Ada contributor can keep using the wonderful Ada built-in dependency management. If he wants to share a library, an appropriately crafted gpr file will expose the code to the C/C++ side in the usual way. If the result is a mere executable things are even simpler.

2) The C/C++ side just keeps working as usual, enabling the Ada parts if needed.

I've prepared a couple of CMake helper macros that give the basic idea and can be enhanced for more involved actions (like installing the Ada library, etc). It is here [1]. Basically, you issue an add_ada_library() and that's it for the CMake side.

Incidentally, I saw another effort to integrate gnat and CMake, but it seems it is going the full CMake way: adding Ada support so source files are recognized and compiled individually and so on. I'm not sure how the binding stage is managed there, but for interested people here it is too [2].

[1] https://github.com/mosteo/ada4cmake

[2] https://github.com/offa/cmake-ada

Setting an Address in a Pure Package?

From: Simon Wright <simon@pushface.org>
Date: Fri, 10 Jul 2015 12:48:29 +0100
Subject: Q: setting an address with pure?
Newsgroups: comp.lang.ada

I need to specify the address at which a hardware object appears.

This works:

```
PIOA : PIO
  with
    Import,
    Convention => Ada,
    Address    =>
          System.Storage_Elements.
          To_Address (16#400E0E00#);
```

but means that I can't declare the package Pure (or even Preelaborate).

Is anyone aware of any GNAT feature that would allow such a package to be Pure? There are similar things in their package Standard, for example Standard'Address_Size.

I tried

```
PIOG : PIO with Import, Convention =>
          Ada;
  for PIOG use at 16#400E1800#;
```

but, besides warning me that 'use at' is obsolescent, GNAT still expects System.Address not universal integer.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 10 Jul 2015 17:37:23 -0500
Subject: Re: setting an address with pure?
Newsgroups: comp.lang.ada

> [...]

You can't have a variable at library level in a Pure package (there shall be no state), so it makes no sense at all for such a package to contain any hardware objects.

I don't see any reason why this can't be Preelaborated (System.Storage_Elements is Pure, after all); that depends on the

initialization of the type PIO (it needs to have "preelaborable_initialization"; use the pragma if in doubt).

From: Rasika Srinivasan <RasikaSrinivasan@gmail.com>
Date: Mon, 13 Jul 2015 16:41:44 -0700
Subject: Re: setting an address with pure?
Newsgroups: comp.lang.ada

Does it have to be pure?

I use

```
 pragma Restrictions
(No_Elaboration_Code);
```

then:

```
 DACMAP : DAC_CR_Type
   with Volatile,
        Address => System'To_Address
             (16#NNNN_NNNN#);
```

I am not sure if this is what you are aiming for but appears to work so for. NNNN_NNNN above is the base address of the DAC map as provided by the STM32 Ref Manual.

From: Simon Wright <simon@pushface.org>
Date: Tue, 14 Jul 2015 08:38:36 +0100
Subject: Re: setting an address with pure?
Newsgroups: comp.lang.ada

> Does it have to be pure?

I don't have any objection to elaboration code per se, I just wanted to get as near to Pure as I could.

I did use that restriction for a Cortex Reset_Handler, which is called by the hardware before any elaboration occurs at all.

> [...] Address => System'To_Address (16#NNNN_NNNN#); [...]

This was just what I wanted, thanks very much!

This is actually in the secret documentation. [...]

http://docs.adacore.com/gnat_rm-docs/html/gnat_rm/gnat_rm/implementation_defined_attributes.html#attribute-to-address

From: Simon Wright <simon@pushface.org>
Date: Mon, 13 Jul 2015 20:50:49 +0100
Subject: Re: Q: setting an address with pure?
Newsgroups: comp.lang.ada

> [...]

> for PIOG'Address use System.Storage_Elements.To_Address(16#400E1C00#);

>

> Since System.Storage_Elements is Pure, that should work fine here. You seemed to indicate that it did not. What's the error message for that?

non-static call not allowed in preelaborated unit

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 13 Jul 2015 14:10:07 -0500
Subject: Re: setting an address with pure?
Newsgroups: comp.lang.ada

> GNAT says that the call to System.Storage_Elements.To_Address is invalid because "non-static call not allowed in preelaborated unit", whether Preelaborable_Initialization is applied or not. I take it that To_Address isn't static because its result type isn't scalar? (4.9(19))

To_Address is a function, not an operator. 4.9(19) does not apply (it only applies to predefined operators, and To_Address is neither an operator nor predefined [language-defined /= predefined]). There is no way to have a static function in Ada to date. (We've been talking about adding an aspect on expression functions to allow that.)

This seems like a bug in Ada to me, it's a consequence of making Address a private type (it's not private in Janus/Ada because we didn't want the incompatibility that would have resulted, so I've never seen this before). It certainly seems wrong that an address clause for a hardware entity can't be used in a preelaborated unit.

> I think this may be a failure in a GNAT extension; To_Address has pragma Pure_Function applied to it, and[2]

>

> "It specifies that the function Entity is to be considered pure for the purposes of code generation. This means that the compiler can assume that there are no side effects, and in particular that two calls with identical arguments produce the same result. It also means that the function can be used in an address clause."

>

> [1] http://www.ada-auth.org/standards/12rm/html/RM-4-9.html#p19

> [2] https://gcc.gnu.org/onlinedocs/gnat_rm/Pragma-Pure_005fFunction.html

Possibly. In "pedantic" mode, the function would have to be an error because implementation-defined stuff shouldn't be changing basic properties (like staticness) of language-defined subprograms. But it certainly makes sense to work-around this language design flaw.

I'd encourage you to post this problem on Ada-Comment, so that it gets on the ARG agenda. (If we end up with a static function aspect, it would make sense to apply it to To_Address.)

From: Mark Lorenzen <mark.lorenzen@gmail.com>
Date: Tue, 14 Jul 2015 02:36:03 -0700
Subject: Re: setting an address with pure?
Newsgroups: comp.lang.ada

> It certainly seems wrong that an address clause for a hardware entity can't be used in a preelaborated unit.

I have been bitten by this restriction several times and it's a real pain.

> I'd encourage you to post this problem on Ada-Comment, so that it gets on the ARG agenda. (If we end up with a static function aspect, it would make sense to apply it to To_Address.)

Yes please - the current restriction prevents many low-level I/O packages from having preelaborable elaboration, which transitively prevents a whole I/O library from having preelaborable elaboration.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 16 Jul 2015 14:14:29 -0500
Subject: Re: Q: setting an address with pure?
Newsgroups: comp.lang.ada

>> I made the type System.Address globally visible as a 32-bit unsigned. That's why the above statements work in AVR-Ada and probably in MSP-Ada.

> Ah! Well if that is permitted by the LRM (and obviously, only for targets where it is valid) it certainly makes life easier!

It's permitted by the RM. There is Implementation Advice that System.Address be a private type, but of course the reason that it is advice is that it might not be appropriate to all implementations. In this particular case, though, its advice mainly because it would have been incompatible on some implementations to make it private (Janus/Ada is in that category); I think the feeling was that new implementations should have it private. (But we now know that's clearly wrong if preelaboration is going to be used with hardware access.)

Hopefully, Ada 202x will find a solution to this problem (and maybe even in the 2018 update), so it's fixed for good rather than depending on the goodness of your RTS implementer (many who are likely to be unaware of this problem).

From: Simon Wright <simon@pushface.org>
Date: Thu, 16 Jul 2015 19:54:54 +0100
Subject: Re: Q: setting an address with pure?
Newsgroups: comp.lang.ada

[...]

See also pragma Allow_Integer_Address[1] - not sure whether it's reached publicly accessible compilers yet.

[1] http://docs.adacore.com/gnat_rm-docs/html/gnat_rm/gnat_rm/implementation_defined_pragmas.html#pragma-allow-integer-address

Promoting Ada

From: Richard Riehle <rriehle@itu.edu>
Date: Wed, 19 Aug 2015 15:19:43 -0700
Subject: Re: If not Ada, what else...
Newsgroups: comp.lang.ada

As to your comment about books, that is why I wrote Ada Distilled, to provide simple, fully coded, tested examples with line-by-line comments. Although the Ada 2012 version is not quite ready (Ed Colbert of Absolute Software is updating it), many people still seem to find the Ada 95 version (which includes some 2005 examples) useful as a place to get started.

You are correct about other things. Greedy compiler and tool vendors (with the exception of Meridian and RR Software) were overpricing the product so few hobbyists or start-ups could afford to choose Ada. Only Meridian provided a fully functional Ada compiler for Windows at a reasonable price. Janus was a really good compiler, but did not have easy support for Windows programming. Alsys was huge, cumbersome, too expensive, and not suitable for any small organisation. The Alsys compiler did generate some pretty good code, but no one was concerned about that. There weren't many other options.

So, community colleges continued to prefer Turbo-Pascal, a product that wowed everyone at the time. I talked with Phillipe Kahn about Ada. He would have loved to have had an opportunity to create a Turbo-Ada, but the timing was wrong, and the opportunity was lost. The one (and perhaps, only) good thing Reifer did when he was in charge at AJPO was fund the initial work on GNAT. Once he left AJPO, he began to publicly disparage Ada, and that did not help at all.

The poorly worded letter from Emmett Page set the stage for Ada's quick demise within the DoD. Now, there is no mandate, and most of the people I know in the DoD software community have interpreted that letter as, not simply cancelling the mandate, but cancelling Ada in favour of anything but Ada. The cancellation of the mandate was a premature and devastating event, occurring exactly at the moment when Ada, as a language design (Ads 95) was poised for extraordinary success.

Ada, as a programming language, is still one of the very best for real engineering of software solutions (not so good for Q&D or hacking), but we have very few engineers in software practice. We have lots of talented programmers, but few of them have any engineering background or understanding of engineering. An interesting outcome of learning Ada, for many of them, was a better understanding of what we really meant by the term, software engineering.

We, the past and present devotees of Ada, have made a lot of mistakes. It is not clear that we can recover from the bad impression so many of our software developer colleagues have regarding Ada. However, the new standard includes some advanced computer science and software engineering features not present in other, if any, software engineering languages: axiomatic program design (Hoare, Dijkstra), predicate calculus expressions, and much more.

We can, perhaps, rescue Ada's reputation, by reaching out beyond our own narrow community with information about these powerful capabilities. That can include more academic papers that use Ada, more articles in places that programmers read, offering to teach an Ada class at local colleges, and using Ada for more applications that real people use.

I am now old, soon to enjoy my 80th birthday. My time as an advocate will soon have passed. Perhaps some of you who are younger can find the course and energy to do something to promote real software engineering using the one language designed to support an engineering approach to software development: Ada. It is, in my view, still Ada. It is certainly not C++. Never has been. Why would anyone choose a language that is inherently error-prone and expect a result that is error-free?

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.

2015

October 08 — 5th **International Workshop on Design, Modeling and Evaluation of Cyber Physical Systems** (CyPhy'2015), Amsterdam, the Netherlands. Topics include: development of industrial or research-oriented cyber-physical systems in domains such as robotics, smart systems (homes, vehicles, buildings), medical and healthcare devices, future generation networks; comparisons of state of the art tools in industrial practice; etc.

October 12-14 — 17th **International System and Design Languages Forum** (SDL'2015), Berlin, Germany. Topics include: industrial application reports (industrial usage reports, standardization activities, tool support and frameworks, domain-specific applicability such as telecommunications, aerospace, automotive, control, ...), model-driven development, evolution of development languages (domain-specific language profiles especially for dependability, modular language design, semantics and evaluation, methodology for application, ...), etc.

October 12-15 — 13th **International Symposium on Automated Technology for Verification and Analysis** (ATVA'2015), Shanghai, China. Topics include: program analysis and software verification; analytical techniques for safety, security, and dependability; testing and runtime analysis based on verification technology; analysis and verification of parallel and concurrent hardware/software systems; verification in industrial practice; applications and case studies; etc.

October 12-15 — 27th **Annual** IEEE **Software Technology Conference** (STC'2015), Long Beach, California, USA. Topics include: critical infrastructure challenges, agile/lean development, affordability, open source, systems engineering challenges for software-intensive systems, etc.

☺ October 13-14 — 11th **European Computer Science Summit** (ECSS'2015), Vienna, Austria. Includes: Public Lecture "Ada Countess of Lovelace - A One-Person Opera, and The Role of Women in Computing".

☺ October 18-21 — 24th **International Conference on Parallel Architectures and Compilation Techniques** (PACT'2015), San Francisco, California, USA. Topics include: parallel architectures and computational models; compilers and tools for parallel computer systems; middleware and run time system support for parallel computing; support for correctness in concurrent hardware and software; parallel programming languages, algorithms and applications; applications and experimental systems studies; etc.

October 21-23 — 18th IEEE **International Conference on Computational Science and Engineering** (CSE'2015), Porto, Portugal. Includes tracks on: scientific and engineering computing; CSE education; embedded and ubiquitous computing; security, privacy and trust; distributed and parallel computing; dependable, reliable and autonomic computing; etc.

☺ 2015/10/21 — **Workshop on Exascale Multi/many Core Computing Systems** (MuCoCoS'2015). Topics include: methods and tools for preparing applications for Exascale; programming models, languages, libraries and compilation techniques; run-time systems; etc.

October 22-23 — 9th ACM/IEEE **International Symposium on Empirical Software Engineering and Measurement** (ESEM'2015), Beijing, China. Topics include: industrial experience and case studies, qualitative methods, replication of empirical studies, empirically-based decision making, evaluation and comparison of techniques and models, quality measurement and assurance, measurement and process

improvement programs, reports on the benefits / costs associated with using certain technologies, software project experience and knowledge management, etc.

October 25-27 ACM SIGPLAN 8th **International Conference on Software Language Engineering** (SLE'2015), Pittsburgh, Pennsylvania, USA. Topics include: techniques for software language reuse, evolution and management of variations (syntactic/semantic) within language families; applications of DSLs for different purposes (incl. modeling, simulating, generation, description, checking); novel applications and/or empirical studies on any aspect of SLE (development, use, deployment, and maintenance of software languages); etc.

☺ October 25-30 ACM **Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2015), Pittsburgh, Pennsylvania, USA. Topics include: all aspects of software construction and delivery, at the intersection of programming, languages, and software engineering.

☺ October 26 6th **Workshop on Evaluation and Usability of Programming Languages and Tools** (PLATEAU'2015). Topics include: methods, metrics and techniques for evaluating the usability of languages and language tools; making programs easier to read, write, and maintain; allowing programmers to write more flexible and powerful programs; restricting programs to make them more safe and secure; empirical studies of programming languages; methodologies and philosophies behind language and tool evaluation; software design metrics and their relations to the underlying language; user studies of language features and software engineering tools; critical comparisons of programming paradigms; tools to support evaluating programming languages; etc.

October 29-31 12th **International Colloquium on Theoretical Aspects of Computing** (ICTAC'2015), Cali, Colombia. Topics include: principles and semantics of programming languages; relationship between software requirements, models and code; program static and dynamic analysis and verification; software specification, refinement, verification and testing; model checking and theorem proving; integration of theories, formal methods and tools for engineering computing systems; models of concurrency, security, and mobility; real-time, embedded, hybrid and cyber-physical systems; etc.

November 02-05 26th IEEE **International Symposium on Software Reliability Engineering** (ISSRE'2015), Washington DC, USA. Topics include: reliability, availability, and safety of software systems; verification and validation; software quality; software security; dependability, fault tolerance, survivability, and resilience of software systems; systems (hardware + software) reliability engineering; etc.

November 03-06 17th **International Conference on Formal Engineering Methods** (ICFEM'2015), Paris, France. Topics include: abstraction and refinement; program analysis; software verification; software model checking; formal methods for object and component systems, concurrent and real-time systems, cyber-physical systems, for software safety, security, reliability and dependability; tool development, integration and experiments involving verified systems; formal methods used in certifying products under international standards; formal model-based development and code generation; etc.

November 04-06 **Symposium on Dependable Software Engineering: Theories, Tools and Applications** (SETTA'2015), Nanjing, 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.

♦ November 05 **High Integrity Software 2015** (HIS'2015), Bristol, UK. Sponsored by AdaCore and Altran.

November 09-13 30th IEEE/ACM **International Conference on Automated Software Engineering** (ASE'2015), Lincoln, Nebraska, USA. Topics include: foundations, techniques and tools for automating the analysis, design, implementation, testing, and maintenance of large software systems, such as open systems development, component-based systems, product line methods, re-engineering, specification languages, maintenance and evolution, software architecture and design, testing, verification, and validation, model-based software development, model transformation, modeling language semantics, etc.

November 15-20 10th **International Conference on Software Engineering Advances** (ICSEA'2015), Barcelona, Spain. Topics include: advances in fundamentals for software development; advanced mechanisms for software development; advanced design tools for developing software; software security, privacy, safeness; specialized software advanced applications; open source software; agile software techniques; software deployment and maintenance; software engineering techniques, metrics, and formalisms; software economics, adoption, and education; improving productivity in research on software engineering; etc.

November 15-20 28th **International Conference for High Performance Computing, Networking, Storage and Analysis** (SC'2015), Austin, Texas, USA.

November 18-20 21st IEEE **Pacific Rim International Symposium on Dependable Computing** (PRDC'2015), Zhangjiajie, China. Topics include: software and hardware reliability, testing, verification, and validation; dependability measurement, modeling, evaluation, and tools; software aging and rejuvenation; safety-critical systems and software; dependability issues in distributed and parallel systems, in real-time systems, in aerospace and embedded systems, in cyber-physical systems, ...; etc.

Nov 30 - Dec 02 13th **Asian Symposium on Programming Languages and Systems** (APLAS'2015), Pohang, Korea. Topics include: foundational and practical issues in programming languages and systems, such as semantics, design of languages and type systems, domain-specific languages, compilers, interpreters, abstract machines, program analysis, verification, model-checking, software security, concurrency and parallelism, tools and environments for programming and implementation, etc.

December 01-04 22nd **Asia-Pacific Software Engineering Conference** (APSEC'2015), New Delhi, India. Theme: "Software Process and Product Engineering". Topics include: embedded real-time systems; formal methods; product-line software engineering; SE environments and tools; security, reliability, and privacy; software architecture and design; software engineering methods; software maintenance and evolution; software process and standards; testing, verification, and validation; etc.

December 02-04 16th **International Conference on Product Focused Software Process Improvement** (PROFES'2015), Bolzano-Bozen, Italy. Topics include: software engineering techniques, methods, and technologies for product-focused software development and process improvement as well as their practical application in industrial settings.

December 08-11 16th ACM/IFIP/USENIX **International Middleware Conference** (Middleware'2015), Vancouver, Canada. Topics include: design, implementation, deployment, and evaluation of distributed system platforms and architectures for computing, storage, and communication environments; reliability and fault-tolerance; real-time solutions; scalability and performance; programming frameworks, parallel programming, and design methodologies for middleware; methodologies and tools for middleware design, implementation, verification, and evaluation; retrospective reviews of middleware paradigms; etc.

♦ Dec 09-10 **Ada Lovelace 200 Symposium: celebrating the life and legacy of Ada Lovelace**. Oxford, UK.

December 09-12 20th **International Conference on Engineering of Complex Computer Systems** (ICECCS'2015), Gold Coast, Australia. Topics include: verification and validation, security and privacy of complex systems, model-driven development, reverse engineering and refactoring, design by contract, agile methods, safety-critical & fault-tolerant architectures, real-time and embedded systems, cyber-physical systems, tools and tool integration, past reflections and future outlooks, industrial case studies, etc.

December 10 200th birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!

December 12-14 7th **International Symposium on Parallel Architectures, Algorithms and Programming** (PAAP'2015), Nanjing, China. Topics include: multi/many-core programming, formal methods and verification, parallel programming languages, parallel compilers and runtime systems, task mapping and job scheduling, secure distributed computing, resource allocation and management, software engineering for parallel/distributed systems, etc.

☺ December 14-17 21st IEEE **International Conference on Parallel and Distributed Systems** (ICPADS'2015), Melbourne, Australia.

2016

January 07-09 17th IEEE **International Symposium on High Assurance Systems Engineering** (HASE'2016), Orlando, Florida, USA. Topics include: tools and techniques used to design and construct systems that, in addition to meeting their functional objectives, are safe, secure, and reliable.

☺ January 18-22 43rd ACM SIGPLAN-SIGACT **Symposium on Principles of Programming Languages** (POPL'2016), St. Petersburg, Florida, USA.

January 19-22 8th **Software Quality Days Conference** (SWQD'2016), Vienna, Austria. Theme: "The Future of Systems and Software Development: Build in Quality & Efficiency right from the Start". Topics include: improvement of software development methods and processes; testing and quality assurance of software and software-intensive systems; domain specific quality issues such as embedded, medical, automotive systems; novel trends in software quality; etc.

February 17-19 24th Euromicro **International Conference on Parallel, Distributed and Network-Based Processing** (PDP'2016), Heraklion, Crete, Greece. Topics include: embedded parallel and distributed systems; multi- and many-core systems; programming languages and environments; runtime support systems; performance prediction and analysis; shared-memory and message-passing systems; dependability and survivability; real-time distributed applications; formal approaches to parallel and distributed systems; security in parallel, distributed and network-based computing; multi-core and many-core systems for embedded computing; etc.

February 19-21 4th **International Conference on Model-Driven Engineering and Software Development** (MODELSWARD'2016), Rome, Italy. Topics include: domain-specific modeling, general-purpose modeling languages and standards, syntax and semantics of modeling languages, model-based testing and validation, model execution and simulation, model quality assurance techniques, component-based software engineering, software factories and software product lines, etc. Deadline for submissions: October 14, 2015 (tutorials, demos, panels), October 29, 2015 (position papers), December 3, 2015 (doctoral consortium).

March 14-17 15th **International Conference on Modularity** (Modularity'2016), Málaga, Spain. Topics include: new modularity mechanisms in programming, modeling, and domain-specific languages; evaluation of modularity mechanisms in case studies; role of modularity in the evolution of software systems; modular re-engineering of legacy code; module (feature) interactions; novel module verification and testing techniques; modularity supported by tools, such as view extraction, visualization, recommendation, and refactoring tools; etc. Deadline for submissions: October 30, 2015 (abstracts, workshops), November 6, 2015 (Research Results papers, Modularity Visions papers), December 5, 2015 (workshop papers).

March 17-18 25th **International Conference on Compiler Construction** (CC'2016), Barcelona, Spain. Topics include: work on processing programs in the most general sense, such as, compilation and interpretation techniques, run-time techniques (memory management, virtual machines, ...), programming tools (refactoring editors, checkers, verifiers, compilers, debuggers, and profilers), techniques for specific domains (secure, parallel, distributed, embedded, ... environments), design and implementation of novel language constructs and programming models. Deadline for submissions: November 13, 2015 (abstracts), November 20, 2015 (papers).

April 02-08 19th **European Joint Conferences on Theory and Practice of Software (**ETAPS'2016), Eindhoven, the Netherlands. 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). Deadline for submissions: October 9, 2015 (abstracts), October 16, 2015 (full papers).

April 04-08 31st ACM **Symposium on Applied Computing** (SAC'2016), Pisa, Italy.

☺ April 04-08 **Track on Programming Languages** (PL'2016). Topics include: compiling techniques, domain-specific languages, garbage collection, language design and implementation, languages for modeling, model-driven development, new programming language ideas

and concepts, practical experiences with programming languages, program analysis and verification, programming languages from all paradigms, etc.

☺ April 04-08 **Track on Multicore Software Engineering, Performance, Applications and Tools** (MUSEPAT'2016). Topics include: software engineering for multicore (CPU or GPU); specification and modeling of multicore systems; programming models, languages, compiler techniques and development tools for multicore; parallel and distributed testing and debugging; evolving sequential software to leverage multicore and manycore hardware; performance and optimization of multicore software; domain- and platform-specific multicore software issues (e.g., issues in scientific computing); etc.

☺ April 04-08 **Track on Object-Oriented Programming Languages and Systems** (OOPS'2016). Topics include: aspects and components; code generation and optimization; distribution and concurrency; formal verification; integration with other paradigms; interoperability, versioning and software evolution and adaptation; language design and implementation; modular and generic programming; runtime verification; secure and dependable software; static analysis; testing and debugging; type systems; etc.

♦ April 11-13 18th **International Real-Time Ada Workshop** (IRTAW'2016), Benicàssim, Spain. In cooperation with Ada-Europe. Deadline for submissions: January 22, 2016 (position papers).

April 27-28 11th **International Conference on Evaluation of Novel Approaches to Software Engineering** (ENASE'2016), Rome, Italy. Topics include: comparing novel approaches with established traditional practices and evaluating them against software quality criteria, software and systems development methodologies, software process improvement, software product line engineering, architectural design and frameworks, software quality management, software change and configuration management, application integration technologies, geographically distributed software engineering, formal methods, model-driven engineering, etc. Deadline for submissions: October 30, 2015 (papers), November 13, 2015 (workshops), November 24, 2015 (special sessions), December 18, 2015 (tutorials, demos, panels), January 5, 2016 (position papers).

☺ May14-22 38th **International Conference on Software Engineering** (ICSE'2016), Austin, Texas, USA. Deadline for submissions: October 10, 2015 (workshop proposals), October 23, 2015 (Software Engineering in Practice papers, Software Engineering Education and Training papers, Software Engineering in Society papers, Visions of 2025 and Beyond papers), November 20, 2015 (technical briefings proposals, Doctoral Symposium proposals, ACM Student Research Competition, demonstrations proposals), January 13, 2016 (posters proposals), January 22, 2016 (workshop papers).

June 01-05 12th **International Conference on integrated Formal Methods** (iFM'2016), Reykjavík, Iceland. Topics include: hybrid approaches to formal modelling and analysis; i.e., the combination of (formal and semi-formal) methods for system development, regarding modelling and analysis, and covering all aspects from language design through verification and analysis techniques to tools and their integration into software engineering practice.

♦June 13-17 21st **International Conference on Reliable Software Technologies - Ada-Europe'2016** Pisa, Italy. Sponsored by Ada-Europe, in cooperation (pending) with ACM SIGAda, SIGBED, SIGPLAN, and the Ada Resource Association (ARA). Deadline for submissions: January 17, 2016 (papers, tutorials, workshops, industrial presentations).

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

The second UK conference on High Integrity Software will take place in Bristol, UK, on 5th November 2015. This one-day event offers the UK's foremost opportunity for engineers to share information about challenges and solutions in the domain of trustworthy software engineering for safety, security and business-critical applications.

This year's conference will feature three keynote speakers. Prof. Ian Phillips, Principal Staff Engineer at ARM, will talk about the role of software in overall system integrity. Prof. Phil Koopman, CMU, will present a study of the Unintended Acceleration (UA) of Toyota vehicles and related software safety issues based on his role as an expert witness. Prof. Mark Little, Vice President Red Hat and CTO of JBoss, will talk about the success of open source software in mission-critical environments and its future role in innovative areas including the Internet of Things.

The programme will also feature technical sessions on software safety, tools & architectures, and threats & security. More details are available on the conference website.

The event includes an exhibition at which vendors will be presenting their tools and services offer for the high integrity software domain. The exhibition will be open during the morning and afternoon breaks, during lunchtime and also during the networking "cocktail hour" at the end of the day.

Attendance at HIS 2015 will cost £175 per delegate, which covers all aspects of this event (breaks, lunches, sessions, exhibition and networking drinks afterwards). Further information and instructions on how to register can be found on the conference website.

www.his-2015.co.uk

SPONSORED BY

AdaCore ALTRAN

Ada Lovelace

Celebrating 200 years of a computer visionary

Ada Lovelace Symposium, Oxford, 2015

An interdisciplinary Symposium celebrating the life and legacy of Ada Lovelace, 1815-1852, will take place at Mathematics Institute, University of Oxford on 9th and 10th December 2015. Ada Lovelace is best known for a remarkable article about Charles Babbage's unbuilt computer, the Analytical Engine, and the symposium will present Lovelace's life and work, in the context of nineteenth century mathematics, science and culture, and present-day thinking on computing and artificial intelligence.

Speakers include: computer scientists John Barnes, Adrian Johnstone, Ursula Martin, Bernard Sufrin and Moshe Vardi; historians of computing and mathematics, June Barrow Green, Elizabeth Bruton, Judith Grabiner, Christopher Hollings and Doron Swade; Lovelace scholars Imogen Forbes-Macphail, Julia Markus and Betty Toole; historian and biographer Richard Holmes; and graphic artist Sydney Padua. Participants in a panel on female icons include computer scientists Valerie Barr and Muffy Calder, founder of Ada Lovelace Day Suw Charman-Anderson, mathematician Cheryl Praeger, and cultural historian Murray Pittock.

A reception and dinner in Balliol College on 9th December includes a pre-dinner address by Lovelace's descendant the Earl of Lytton, and an after dinner speech by philanthropist Dame Stephanie Shirley.

Registration for the symposium is £40, or £90 including the symposium dinner. Some sponsored places are available. For further information and registration see https://blogs.bodleian.ox.ac.uk/adalovelace/

A display at Oxford's Bodleian Library, 13th October – 18th December, includes Lovelace's exercise books, childhood letters, correspondence with Charles Babbage, a newly found daguerreotype, and a new archive discovery showing computational thinking in action –Lovelace, Babbage, magic squares and networks.

Sponsors This event has been made possible thanks to generous sponsorship from ACM, AHRC, British Computer Society, Clay Mathematics Institute, EPSRC, google, IMA, London Mathematical Society, and Queen Mary University of London's cs4fn project.

Professor Ursula Martin CBE
Chair, Ada Lovelace Celebration 2015
University of Oxford
Ursula.Martin@cs.ox.ac.uk

18th International Real-Time Ada Workshop – IRTAW 2016

Hotel Voramar, Benicàssim, Spain
11-13th April 2016

http://www.ada-europe.org/irtaw2016

Call for Papers

The International Real-Time Ada Workshop series has provided a forum for identifying issues with real-time system support in Ada and for exploring possible approaches and solutions, and has attracted participation from key members of the research, user, and implementer communities worldwide. Recent International Real-Time Ada Workshop meetings contributed to the Ada 2005/Ada 2012 standards, especially with respect to the tasking features, the real-time and high-integrity systems annexes, and the standardization of the Ravenscar Tasking Profile.

In keeping with this tradition, the goals of IRTAW-18 will be to:

- Review Ada 2012 Issues vis-a-vis real-time systems;
- Examine experiences in the use of Ada 2012 for real-time systems and applications;
- Implementation approaches for Ada 2012 real-time features;
- Consider developing other real-time Ada profiles in addition to the Ravenscar profile;
- Analyze the implications to Ada with multiprocessors in development of real-time systems;
- Investigate paradigms for using Ada for real-time distributed systems, with special emphasis on robustness as well as hard, flexible and application-defined scheduling;
- Analyse specific patterns and libraries for real-time systems development in Ada;
- Evaluate Ada in context of the certification of safety-critical and/or security-critical real-time systems;
- Examine the Real-Time Specification for Java and other languages for real-time systems development, their current implementations and their interoperability with Ada in embedded real-time systems;
- Investigate industrial experience with Ada and the Ravenscar Profile in real-time projects;
- Consider the language vulnerabilities of the Ravenscar and full language definitions;
- Consider testing for compliance with the Real-Time Annex.

Participation at IRTAW-18 is by invitation following the submission of a position paper addressing one or more of the above topics or related real-time Ada issues. Alternatively, anyone wishing to receive an invitation, but for one reason or another is unable to produce a position paper, may send in a one-page position statement indicating their interests. Priority will be given to submitted papers.

Submission Requirements

Position papers should not exceed ten pages in typical IEEE conference layout, excluding code inserts. All accepted papers will appear, in their final form, in the Workshop Proceedings, which will be published as a special issue of Ada Letters (ACM Press). Selected papers will also appear in the Ada User Journal. Authors with a relevant paper submitted to the 21st International Conference on Reliable Software Technologies – Ada-Europe 2016 (deadline 17 January, 2016) may offer an extended abstract of the same material to IRTAW 18. Please submit position papers, in PDF format, to the Program Chair by e-mail: *stephen.michell@maurya.on.ca*

Important Dates

Paper Submission: **22 January, 2016**
Notification of Acceptance: 19 February, 2016
Confirmation of Attendance: 4 March, 2016
Final Paper Due: 25 March, 2016
Workshop: April 11-13, 2016

Program Chair
Stephen Michell, *Maurya Software Inc, Canada*

Workshop Chair
Jorge Real, *Universitat Politècnica de València*

AdaCore The GNAT Pro Company

Ada europe

DISCA DEPARTAMENT D'INFORMÀTICA DE SISTEMES I COMPUTADORS

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Conference Chair

Giorgio Buttazzo
Scuola Superiore Sant'Anna

Program Co-Chairs

Marko Bertogna
Univ. of Modena and Reggio Emilia

Luís Miguel Pinho
CISTER Research Centre/ISEP

Special Session Chair

Eduardo Quiñones
Barcelona Supercomputing Center

Tutorial and Workshop Chair

Jorge Real
Universitat Politècnica de València

Industrial Co-Chairs

Marco Di Natale
Scuola Superiore Sant'Anna

Tullio Vardanega
Università di Padova

Publication Chair

Geoffrey Nelissen
CISTER Research Centre/ISEP

Exhibition Co-Chairs

Paolo Gai
Evidence Srl

Ahlan Marriot
White Elephant GmbH

Publicity Co-Chairs

Mauro Marinoni
Scuola Superiore Sant'Anna

Dirk Craeynest
Ada-Belgium & KU Leuven

Local Chair

Ettore Ricciardi
ISTI-CNR, Pisa

General Information

The **21st International Conference on Reliable Software Technologies – Ada-Europe 2016** will take place in Pisa, Italy. 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.

Schedule

17 January 2016	Submission of papers, industrial presentation, tutorial and workshop proposals.
10 March 2016	Notification of acceptance to all authors
24 March 2016	Camera-ready version of papers required
2 May 2016	Industrial presentations, tutorial and workshop material required

Topics

The conference has over the years become a leading international forum for providers, practitioners and researchers in reliable software technologies. The conference presentations will illustrate current work in the theory and practice of the design, development and maintenance of long-lived, high-quality software systems for a challenging variety of application domains. The program will allow ample time for keynotes, Q&A sessions and discussions, and social events. Participants include practitioners and researchers representing industry, academia and government organizations active in the promotion and development of reliable software technologies.

This edition of Ada-Europe features a focused **Special Session on Safe, Predictable Parallel Software Technologies**. Following the increasing trend of usage of Multi-/Many-core systems, it is more and more important to assess how reliable software technologies need to adapt to these complex platforms, as well as how parallel models need to adapt to domains in which safety and predictability is a must. Topics include (but are not limited to): **Predictable Parallel Programming Models, Compiler Support for Parallel Execution, Parallel Runtimes, Automatic Parallelization, Safety Issues and Reliability Mechanisms for Parallel Execution, Software Modelling and Design Approaches**.

For the **general track of the conference**, topics of interest include but are not limited to (full list on the website): Real-Time and Embedded Systems, Mixed-Criticality Systems, Theory and Practice of High-Integrity Systems, Software Architectures, Methods and Techniques for Software Development and Maintenance, Software Quality, Mainstream and Emerging Applications, Experience Reports in Reliable System Development, Experiences with Ada.

Call for Regular and Special Session Papers

Authors of papers which are to undergo peer review for acceptance are invited to submit original contributions by 17 January 2016. Paper submissions shall not exceed 14 LNCS-style pages in length. Authors for both the general track and the special session shall submit their work via EasyChair at https://easychair.org/conferences/?conf=adaeurope2016. The format for submission is solely PDF.

Proceedings

The conference proceedings will be published in the Lecture Notes in Computer Science (LNCS) series by Springer, and will be available at the conference. The authors of accepted regular and special session papers shall prepare camera-ready submissions in full conformance with the LNCS style, not exceeding 14 pages and strictly by 24 March 2016. For format and style guidelines authors should refer to http://www.springer.de/comp/lncs/authors.html. Failure to comply and to register for the conference by that date will prevent the paper from appearing in the proceedings.

The International Conference on Reliable Software Technologies is ranked class A in the CORE ranking and Microsoft Academic Search has it in the top third for conferences on programming languages. The conference is listed in DBLP, SCOPUS and Web of Science Conference Proceedings Citation index, among others.

Awards

Ada-Europe will offer honorary awards for the best regular paper and the best presentation.

Call for Industrial Presentations

The conference seeks industrial presentations which deliver value and insight but may not fit the selection process for regular papers. Authors are invited to submit a presentation outline of exactly 1 page in length by 17 January 2016. Submissions shall be made via EasyChair following the link https://easychair.org/conferences/?conf=adaeurope2016. The format for submission is solely PDF.

The Industrial Committee will review the submissions and make the selection. The authors of selected presentations shall prepare a final short abstract and submit it by 2 May 2016, aiming at a 20-minute talk. The authors of accepted presentations will be invited to submit corresponding articles for publication in the *Ada User Journal* (http://www.ada-europe.org/auj/), which will host the proceedings of the Industrial Program of the Conference. For any further information please contact the Industrial Co-chairs directly.

Call for Tutorials

Tutorials should address subjects that fall within the scope of the conference and may be proposed as either half- or full-day events. Proposals should include a title, an abstract, a description of the topic, a detailed outline of the presentation, a description of the presenter's lecturing expertise in general and with the proposed topic in particular, the proposed duration (half day or full day), the intended level of the tutorial (introductory, intermediate, or advanced), the recommended audience experience and background, and a statement of the reasons for attending. Proposals should be submitted by e-mail to the Tutorial Chair. The authors of accepted full-day tutorials will receive a complimentary conference registration as well as a fee for every paying participant in excess of 5; for half-day tutorials, these benefits will be accordingly halved. The *Ada User Journal* (http://www.ada-europe.org/auj/) will offer space for the publication of summaries of the accepted tutorials.

Call for Workshops

Workshops on themes that fall within the conference scope may be proposed. Proposals may be submitted for half- or full-day events, to be scheduled at either end of the conference week. Workshop proposals should be submitted to the Tutorial and Workshop Chair. The workshop organizer shall also commit to preparing proceedings for timely publication in the *Ada User Journal* (http://www.ada-europe.org/auj/).

Call for Exhibitors

The commercial exhibition will span the three days of the main conference. Vendors and providers of software products and services should contact the Exhibition Chair for information and for allowing suitable planning of the exhibition space and time.

Grants for Reduced Student Fees

A limited number of sponsored grants for reduced fees is expected to be available for students who would like to attend the conference or tutorials. Contact the Conference Chair for details.

Venue

The conference will take place at Scuola Superiore Sant'Anna (left images, including the aula magna where the main conference sessions will take place), in the heart of Pisa, Italy. June is full of events in Pisa, including in the conference week the Saint Patron's festivities (San Ranieri) with the Luminara on the night of June 16 (thousands of candles burn and reflect on the river – image on the right). Plan in advance! It is absolutely worth it!

Proven Test Solutions for Reliable Embedded Software

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

-- Honeywell

VECTOR software

vectorcast.com

VectorCAST is a TÜV SÜD Certified Software Tool for Safety Related Development

Vector Software, Inc.
Golden Cross House | 8 Duncannon Street | London WC2N 4JF UK | +44 203 603 0120 | sales@vectorcast.com

1842 Notes to the translation of the Sketch of the Analytical Engine

A. A. Lovelace

Editor note: *This article presents the 1842 Notes by Ada Lovelace to the English translation of the article "Sketch of the Analytical Engine invented by Charles Babbage" by L. F. Menabrea, Bibliothèque Universelle de Génève, Nº 82, October 1842* [1]*. The translation and the notes were published in "Taylor's Scientific Memoirs", London, vol. III, 1843, pp. 666-731* [2]*. As the article focuses on the notes themselves, they are all presented separated from the analytical engine description, which is nevertheless provided in preamble for context.*

[1] *The translation and notes are from the online copy at http://www.fourmilab.ch/babbage/sketch.html. Last accessed September 2015.*

[2] *The reader may find an online digitalized copy of this volume of the "Taylor's Scientific Memoirs", in Google Books at https://books.google.pt/books?id=qsY-AAAAYAAJ. Last accessed September 2015.*

Preamble - Sketch of the Analytical Engine Invented by Charles Babbage, by L. F. Menabrea, translation by A. A. Lovelace.

Those labours which belong to the various branches of the mathematical sciences, although on first consideration they seem to be the exclusive province of intellect, may, nevertheless, be divided into two distinct sections; one of which may be called the mechanical, because it is subjected to precise and invariable laws, that are capable of being expressed by means of the operations of matter; while the other, demanding the intervention of reasoning, belongs more specially to the domain of the understanding. This admitted, we may propose to execute, by means of machinery, the mechanical branch of these labours, reserving for pure intellect that which depends on the reasoning faculties. Thus the rigid exactness of those laws which regulate numerical calculations must frequently have suggested the employment of material instruments, either for executing the whole of such calculations or for abridging them; and thence have arisen several inventions having this object in view, but which have in general but partially attained it. For instance, the much-admired machine of Pascal is now simply an object of curiosity, which, whilst it displays the powerful intellect of its inventor, is yet of little utility in itself. Its powers extended no further than the execution of the first four[1] operations of arithmetic, and indeed were in reality confined to that of the first two, since multiplication and division were the result of a series of additions and subtractions. The chief drawback hitherto on most of such machines is, that they require the continual intervention of a human agent to regulate their movements, and thence arises a source of errors; so that, if their use has not become general for large numerical calculations, it is because they have not in fact resolved the double problem which the question presents, that of correctness in the results, united with economy of time.

Struck with similar reflections, Mr. Babbage has devoted some years to the realization of a gigantic idea. He proposed to himself nothing less than the construction of a machine capable of executing not merely arithmetical calculations, but even all those of analysis, if their laws are known. The imagination is at first astounded at the idea of such an undertaking; but the more calm reflection we bestow on it, the less impossible does success appear, and it is felt that it may depend on the discovery of some principle so general, that, if applied to machinery, the latter may be capable of mechanically translating the operations which may be indicated to it by algebraical notation. The illustrious inventor having been kind enough to communicate to me some of his views on this subject during a visit he made at Turin, I have, with his approbation, thrown together the impressions they have left on my mind. But the reader must not expect to find a description of Mr. Babbage's engine; the comprehension of this would entail studies of much length; and I shall endeavour merely to give an insight into the end proposed, and to develop the principles on which its attainment depends.

I must first premise that this engine is entirely different from that of which there is a notice in the 'Treatise on the Economy of Machinery,' by the same author. But as the latter gave rise[2] to the

[1] This remark seems to require further comment, since it is in some degree calculated to strike the mind as being at variance with the subsequent passage, where it is explained that *an engine which can effect these four* operations can in fact effect *every species of calculation*. The apparent discrepancy is stronger too in the translation than in the original, owing to its being impossible to render precisely into the English tongue all the niceties of distinction which the French idiom happens to admit of in the phrases used for the two passages we refer to. The explanation lies in this: that in the one case the execution of these four operations is the *fundamental starting-point*, and the object proposed for attainment by the machine is the *subsequent combination of these* in every possible variety; whereas in the other case the execution of some *one* of these four operations, selected at pleasure, is the *ultimatum*, the sole and utmost result that can be proposed for attainment by the machine referred to, and which result it cannot any further combine or work upon. The one *begins* where the other *ends*. Should this distinction not now appear perfectly clear, it will become so on perusing the rest of the Memoir, and the Notes that are appended to it. —NOTE BY TRANSLATOR.

[2] The idea that the one engine is the offspring and has grown out of the other, is an exceedingly natural and plausible supposition, until reflection reminds us that no *necessary* sequence and connexion need exist between two such inventions, and that they may be wholly independent. M. Menabrea has shared this idea in common with persons who have not his profound and accurate insight into the nature of either engine. In Note A. (see the Notes at the end of the Memoir) it will be found sufficiently explained, however, that this supposition is unfounded. M. Menabrea's

Table I

	A Column of Square Numbers	B First Differences	C Second Differences
	1		
	.....	3	
	4	.....	2 *b*
	.....	5	
a	9	.....	2 *d*
	.....	7	
c	16	.....	2
	.....	9	
	25	.....	2
	.....	11	
	36		

idea of the engine in question, I consider it will be a useful preliminary briefly to recall what were Mr. Babbage's first essays, and also the circumstances in which they originated.

It is well known that the French government, wishing to promote the extension of the decimal system, had ordered the construction of logarithmical and trigonometrical tables of enormous extent. M. de Prony, who had been entrusted with the direction of this undertaking, divided it into three sections, to each of which was appointed a special class of persons. In the first section the formulæ were so combined as to render them subservient to the purposes of numerical calculation; in the second, these same formulæ were calculated for values of the variable, selected at certain successive distances; and under the third section, comprising about eighty individuals, who were most of them only acquainted with the first two rules of arithmetic, the values which were intermediate to those calculated by the second section were interpolated by means of simple additions and subtractions.

An undertaking similar to that just mentioned having been entered upon in England, Mr. Babbage conceived that the operations performed under the third section might be executed by a machine; and this idea he realized by means of mechanism, which has been in part put together, and to which the name Difference Engine is applicable, on account of the principle upon which its construction is founded. To give some notion of this, it will suffice to consider the series of whole square numbers, 1, 4, 9, 16, 25, 36, 49, 64, &c. By subtracting each of these from the succeeding one, we obtain a new series, which we will name the Series of First Differences, consisting of the numbers 3, 5, 7, 9, 11, 13, 15, &c. On subtracting from each of these the preceding one, we obtain the Second Differences, which are all constant and equal to 2. We may represent this succession of operations, and their results, in table I.

From the mode in which the last two columns B and C have been formed, it is easy to see, that if, for instance, we desire to pass from the number 5 to the succeeding one 7, we must add to the former the constant difference 2; similarly, if from the square number 9 we would pass to the following one 16, we must add to the former the difference 7, which difference is in other words the preceding difference 5, plus the constant difference 2; or again, which comes to the same thing, to obtain 16 we have only to add together the three numbers 2, 5, 9, placed obliquely in the direction *ab*. Similarly, we obtain the number 25 by summing up the three numbers placed in the oblique direction *dc*: commencing by the addition 2+7, we have the first difference 9 consecutively to 7; adding 16 to the 9 we have the square 25. We see then that the three numbers 2, 5, 9 being given, the whole series of successive square numbers, and that of their first differences likewise may be obtained by means of simple additions.

Now, to conceive how these operations may be reproduced by a machine, suppose the latter to have three dials, designated as A, B, C, on each of which are traced, say a thousand divisions, by way of example, over which a needle shall pass. The two dials, C, B, shall have in addition a registering hammer, which is to give a number of strokes equal to that of the divisions indicated by the needle. For each stroke of the registering hammer of the dial C, the needle B shall advance one division; similarly, the needle A shall advance one division for every stroke of the registering hammer of the dial B. Such is the general disposition of the mechanism.

This being understood, let us, at the beginning of the series of operations we wish to execute, place the needle C on the division 2, the needle B on the division 5, and the needle A on the division 9. Let us allow the hammer of the dial C to strike; it will strike twice, and at the same time the needle B will pass over two divisions. The latter will then indicate the number 7, which succeeds the number 5 in the column of first differences. If we now permit the hammer of the dial B to strike in its turn, it will strike seven times, during which the needle A will advance seven divisions; these added to the nine already marked by it will give the number 16, which is the square number consecutive to 9. If we now recommence these operations, beginning with the needle C, which is always to be left on the division 2, we shall perceive that by repeating them indefinitely, we may successively reproduce the series of whole square numbers by means of a very simple mechanism.

The theorem on which is based the construction of the machine we have just been describing, is a particular case of the following more general theorem: that if in any polynomial whatever, the highest power of whose variable is *m*, this same variable be increased by equal degrees; the corresponding values of the polynomial then calculated, and the first, second, third, &c. differences of these be taken (as for the preceding series of squares); the *m*th differences will all be equal to each other. So that, in order to reproduce the series of values of the polynomial by means of a machine analogous to the one above described, it is sufficient that there be (*m*+1) dials, having the mutual relations we have indicated. As the differences may be either positive or negative, the machine will have a contrivance for either advancing or retrograding each needle, according as the number to be algebraically added may have the sign *plus* or *minus*.

If from a polynomial we pass to a series having an infinite number of terms, arranged according to the ascending powers of the variable, it would at first appear, that in order to apply the machine to the calculation of the function represented by such a series, the mechanism must include an infinite number of dials, which would in fact render the thing impossible. But in many cases the difficulty will disappear, if we observe that for a great

opportunities were by no means such as could be adequate to afford him information on a point like this, which would be naturally and almost unconsciously *assumed*, and would scarcely suggest any inquiry with reference to it.—NOTE BY TRANSLATOR.

number of functions the series which represent them may be rendered convergent; so that, according to the degree of approximation desired, we may limit ourselves to the calculation of a certain number of terms of the series, neglecting the rest. By this method the question is reduced to the primitive case of a finite polynomial. It is thus that we can calculate the succession of the logarithms of numbers. But since, in this particular instance, the terms which had been originally neglected receive increments in a ratio so continually increasing for equal increments of the variable, that the degree of approximation required would ultimately be affected, it is necessary, at certain intervals, to calculate the value of the function by different methods, and then respectively to use the results thus obtained, as data whence to deduce, by means of the machine, the other intermediate values. We see that the machine here performs the office of the third section of calculators mentioned in describing the tables computed by order of the French government, and that the end originally proposed is thus fulfilled by it.

Such is the nature of the first machine which Mr. Babbage conceived. We see that its use is confined to cases where the numbers required are such as can be obtained by means of simple additions or subtractions; that the machine is, so to speak, merely the expression of one particular theorem [3]of analysis; and that, in short, its operations cannot be extended so as to embrace the solution of an infinity of other questions included within the domain of mathematical analysis. It was while contemplating the vast field which yet remained to be traversed, that Mr. Babbage, renouncing his original essays, conceived the plan of another system of mechanism whose operations should themselves possess all the generality of algebraical notation, and which, on this account, he denominates the *Analytical Engine*.

Having now explained the state of the question, it is time for me to develop the principle on which is based the construction of this latter machine. When analysis is employed for the solution of any problem, there are usually two classes of operations to execute: first, the numerical calculation of the various coefficients; and secondly, their distribution in relation to the quantities affected by them. If, for example, we have to obtain the product of two binomials $(a+bx)$ $(m+nx)$, the result will be represented by am + $(an + bm)$ $x + bnx2$, in which expression we must first calculate am, an, bm, bn; then take the sum of $an + bm$; and lastly, respectively distribute the coefficients thus obtained amongst the powers of the variable. In order to reproduce these operations by means of a machine, the latter must therefore possess two distinct sets of powers: first, that of executing numerical calculations; secondly, that of rightly distributing the values so obtained.

But if human intervention were necessary for directing each of these partial operations, nothing would be gained under the heads of correctness and economy of time; the machine must therefore have the additional requisite of executing by itself all the successive operations required for the solution of a problem proposed to it, when once the *primitive numerical data* for this same problem have been introduced. Therefore, since, from the moment that the nature of the calculation to be executed or of the problem to be resolved have been indicated to it, the machine is, by its own intrinsic power, of itself to go through all the intermediate operations which lead to the proposed result, it must exclude all methods of trial and guess-work, and can only admit the direct processes of calculation[4].

It is necessarily thus; for the machine is not a thinking being, but simply an automaton which acts according to the laws imposed upon it. This being fundamental, one of the earliest researches its author had to undertake, was that of finding means for effecting the division of one number by another without using the method of guessing indicated by the usual rules of arithmetic. The difficulties of effecting this combination were far from being among the least; but upon it depended the success of every other. Under the impossibility of my here explaining the process through which this end is attained, we must limit ourselves to admitting that the first four operations of arithmetic, that is addition, subtraction, multiplication and division, can be performed in a direct manner through the intervention of the machine. This granted, the machine is hence capable of performing every species of numerical calculation, for all such calculations ultimately resolve themselves into the four operations we have just named. To conceive how the machine can now go through its functions according to the laws laid down, we will begin by giving an idea of the manner in which it materially represents numbers.

Let us conceive a pile or vertical column consisting of an indefinite number of circular discs, all pierced through their centres by a common axis, around which each of them can take an independent rotatory movement. If round the edge of each of these discs are written the ten figures which constitute our numerical alphabet, we may then, by arranging a series of these figures in the same vertical line, express in this manner any number whatever. It is sufficient for this purpose that the first disc represent units, the second tens, the third hundreds, and so on. When two numbers have been thus written on two distinct columns, we may propose to combine them arithmetically with each other, and to obtain the result on a third column. In general, if we have a series of columns[5] consisting of discs, which columns we will designate as V0, V1, V2, V3, V4, &c., we may require, for instance, to divide the number written on the column V1 by that on the column V4, and to obtain the result on the column V7. To effect this operation, we must impart to the machine two distinct arrangements; through the first it is prepared for executing a division, and through the second the columns it is to operate on are indicated to it, and also the column on which the result is to be represented. If this division is to be followed, for example, by the addition of two numbers taken on other columns, the two original arrangements of the machine must be simultaneously altered. If, on the contrary, a series of operations of the same nature is to be gone through, then the first of the original arrangements will remain, and the second alone must be altered. Therefore, the arrangements that may be communicated to the various parts of the machine may be distinguished into two principal classes:

[3] See Note A in "Notes by the Translator" section

[4] This must not be understood in too unqualified a manner. The engine is capable under certain circumstances, of feeling about to discover which of two or more possible contingencies has occurred, and of then shaping its future course accordingly. —NOTE BY TRANSLATOR.

[5] See Note B in Section "Notes by the Translator"

First, that relative to the *Operations.*
Secondly, that relative to the *Variables*.

By this latter we mean that which indicates the columns to be operated on. As for the operations themselves, they are executed by a special apparatus, which is designated by the name of *mill*, and which itself contains a certain number of columns, similar to those of the Variables. When two numbers are to be combined together, the machine commences by effacing them from the columns where they are written, that is, it places zero[6] on every disc of the two vertical lines on which the numbers were represented; and it transfers the numbers to the mill. There, the apparatus having been disposed suitably for the required operation, this latter is effected, and, when completed, the result itself is transferred to the column of Variables which shall have been indicated. Thus the mill is that portion of the machine which works, and the columns of Variables constitute that where the results are represented and arranged. After the preceding explanations, we may perceive that all fractional and irrational results will be represented in decimal fractions. Supposing each column to have forty discs, this extension will be sufficient for all degrees of approximation generally required.

It will now be inquired how the machine can of itself, and without having recourse to the hand of man, assume the successive dispositions suited to the operations. The solution of this problem has been taken from Jacquard's apparatus[7], used for the manufacture of brocaded stuffs, in the following manner.

Two species of threads are usually distinguished in woven stuffs; one is the *warp* or longitudinal thread, the other the *woof* or transverse thread, which is conveyed by the instrument called the shuttle, and which crosses the longitudinal thread or warp. When a brocaded stuff is required, it is necessary in turn to prevent certain threads from crossing the woof, and this according to a succession which is determined by the nature of the design that is to be reproduced. Formerly this process was lengthy and difficult, and it was requisite that the workman, by attending to the design which he was to copy, should himself regulate the movements the threads were to take. Thence arose the high price of this description of stuffs, especially if threads of various colours entered into the fabric. To simplify this manufacture, Jacquard devised the plan of connecting each group of threads that were to act together, with a distinct lever belonging exclusively to that group. All these levers terminate in rods, which are united together in one bundle, having usually the form of a parallelopiped with a rectangular base. The rods are cylindrical, and are separated from each other by small intervals. The process of raising the threads is thus resolved into that of moving these various lever-arms in the requisite order. To effect this, a rectangular sheet of pasteboard is taken, somewhat larger in size than a section of the bundle of lever-arms. If this sheet be applied to the base of the bundle, and an advancing motion be then communicated to the pasteboard, this latter will move with it all the rods of the bundle, and consequently the threads that are connected with each of them. But if the pasteboard, instead of being plain, were pierced with holes corresponding to the extremities of the levers which meet it, then, since each of the levers would pass through the pasteboard during the motion of the latter, they would all remain in their places. We thus see that it is easy so to determine the position of the holes in the pasteboard, that, at any given moment, there shall be a certain number of levers, and consequently of parcels of threads, raised, while the rest remain where they were. Supposing this process is successively repeated according to a law indicated by the pattern to be executed, we perceive that this pattern may be reproduced on the stuff. For this purpose we need merely compose a series of cards according to the law required, and arrange them in suitable order one after the other; then, by causing them to pass over a polygonal beam which is so connected as to turn a new face for every stroke of the shuttle, which face shall then be impelled parallelly to itself against the bundle of lever-arms, the operation of raising the threads will be regularly performed. Thus we see that brocaded tissues may be manufactured with a precision and rapidity formerly difficult to obtain.

Arrangements analogous to those just described have been introduced into the Analytical Engine. It contains two principal species of cards: first, Operation cards, by means of which the parts of the machine are so disposed as to execute any determinate series of operations, such as additions, subtractions, multiplications, and divisions; secondly, cards of the Variables, which indicate to the machine the columns on which the results are to be represented. The cards, when put in motion, successively arrange the various portions of the machine according to the nature of the processes that are to be effected, and the machine at the same time executes these processes by means of the various pieces of mechanism of which it is constituted.

In order more perfectly to conceive the thing, let us select as an example the resolution of two equations of the first degree with two unknown quantities. Let the following be the two equations, in which *x* and *y* are the unknown quantities:

$$\begin{cases} mx + ny = d \\ m'x + n'y = d'. \end{cases}$$

We deduce $x = \dfrac{dn' - d'n}{n'm - nm'}$, and for y an analogous expression. Let us continue to represent by V0, V1, V2, &c. the different columns which contain the numbers, and let us suppose that the first eight columns have been chosen for expressing on them the numbers represented *by m, n, d, m', n', d', n* and *n'*, which implies that V0=*m*, V1=*n*, V2=*d*, V3=*m*', V4=*n'*, V5=*d*', V6=*n*, V7=*n*'.

The series of operations commanded by the cards, and the results obtained, may be represented in table II. Since the cards do nothing but indicate in what manner and on what columns the machine shall act, it is clear that we must still, in every particular case, introduce the numerical data for the calculation. Thus, in the example we have selected, we must previously inscribe the numerical values of *m*, *n*, *d*, *m*', *n*', *d*', in the order and on the columns indicated, after which the machine when put in action will give the value of the unknown quantity *x* for this particular case. To obtain the value of *y*, another series of operations analogous to the preceding must be performed. But we see that they will be only four in number, since the denominator of the expression for *y*, excepting the sign, is the same as that for *x*, and

[6] Zero is not *always* substituted when a number is transferred to the mill. This is explained further on in the memoir, and still more fully in Note D. —NOTE BY TRANSLATOR.

[7] See Note C in Section "Notes by the Translator".

Table II

Number of the operations	Operation-cards	Cards of the variables		Progress of the operations
	Symbols indicating the nature of the operations	Columns on which operations are to be performed	Columns which receive results of operations	
1	×	$V_2 \times V_4 =$	$V_8 \ldots\ldots$	$= dn'$
2	×	$V_5 \times V_1 =$	$V_9 \ldots\ldots$	$= d'n$
3	×	$V_4 \times V_0 =$	$V_{10} \ldots\ldots$	$= n'm$
4	×	$V_1 \times V_3 =$	$V_{11} \ldots\ldots$	$= nm'$
5	−	$V_8 - V_9 =$	$V_{12} \ldots\ldots$	$= dn' - d'n$
6	−	$V_{10} - V_{11} =$	$V_{13} \ldots\ldots$	$= n'm - nm'$
7	÷	$\frac{V_{12}}{V_{13}} =$	$V_{14} \ldots\ldots$	$= x = \frac{dn' - d'n}{n'm - nm'}$

equal to *n*'*m*-*nm*'. In the preceding table it will be remarked that the column for operations indicates four successive *multiplications*, two *subtractions*, and one *division*. Therefore, if desired, we need only use three operation-cards; to manage which, it is sufficient to introduce into the machine an apparatus which shall, after the first multiplication, for instance, retain the card which relates to this operation, and not allow it to advance so as to be replaced by another one, until after this same operation shall have been four times repeated. In the preceding example we have seen, that to find the value of *x* we must begin by writing the coefficients *m*, *n*, *d*, *m*', *n*', *d*', upon eight columns, thus repeating *n* and *n*' twice. According to the same method, if it were required to calculate *y* likewise, these coefficients must be written on twelve different columns. But it is possible to simplify this process, and thus to diminish the chances of errors, which chances are greater, the larger the number of the quantities that have to be inscribed previous to setting the machine in action. To understand this simplification, we must remember that every number written on a column must, in order to be arithmetically combined with another number, be effaced from the column on which it is, and transferred to the *mill*. Thus, in the example we have discussed, we will take the two coefficients *m* and *n*', which are each of them to enter into *two* different products, that is *m* into *mn*' and *md*', *n*' into *mn*' and *n*'*d*. These coefficients will be inscribed on the columns V_0 and V_4. If we commence the series of operations by the product of *m* into *n*', these numbers will be effaced from the columns V_0 and V_4, that they may be transferred to the mill, which will multiply them into each other, and will then command the machine to represent the result, say on the column V_6. But as these numbers are each to be used again in another operation, they must again be inscribed somewhere; therefore, while the mill is working out their product, the machine will inscribe them anew on any two columns that may be indicated to it through the cards; and as, in the actual case, there is no reason why they should not resume their former places, we will suppose them again inscribed on V_0 and V_4, whence in short they would not finally disappear, to be reproduced no more, until they should have gone through all the combinations in which they might have to be used.

We see, then, that the whole assemblage of operations requisite for resolving the two above equations[8] of the first degree may be definitely represented in table III.

In order to diminish to the utmost the chances of error in inscribing the numerical data of the problem, they are successively placed on one of the columns of the mill; then, by means of cards arranged for this purpose, these same numbers are caused to arrange themselves on the requisite columns, without the operator having to give his attention to it; so that his undivided mind may be applied to the simple inscription of these same numbers. According to what has now been explained, we see that the collection of columns of Variables may be regarded as a *store* of numbers, accumulated there by the mill, and which, obeying the orders transmitted to the machine by means of the cards, pass alternately from the mill to the store and from the store to the mill, that they may undergo the transformations demanded by the nature of the calculation to be performed.

Hitherto no mention has been made of the *signs* in the results, and the machine would be far from perfect were it incapable of expressing and combining amongst each other positive and negative quantities. To accomplish this end, there is, above every column, both of the mill and of the store, a disc, similar to the discs of which the columns themselves consist. According as the digit on this disc is even or uneven, the number inscribed on the corresponding column below it will be considered as positive or negative. This granted, we may, in the following manner, conceive how the signs can be algebraically combined in the machine. When a number is to be transferred from the store to the mill, and *vice versâ*, it will always be transferred with its sign, which will effected by means of the cards, as has been explained in what precedes. Let any two numbers then, on which we are to operate arithmetically, be placed in the mill with their respective signs. Suppose that we are first to add them together; the operation-cards will command the addition: if the two numbers be of the same sign, one of the two will be entirely effaced from where it was inscribed, and will go to add itself on the column which contains the other number; the machine will, during this operation, be able, by means of a certain apparatus, to prevent any movement in the disc of signs which belongs to the column on which the addition is made, and thus the result will remain with the sign which the two given numbers originally had. When two numbers have two different signs, the addition commanded by the

[8] See Note D in section "Notes by the translator"

Table III

Columns on which are inscribed the primitive data	Number of the operations	Cards of the operations: No. of the Operation-cards	Cards of the operations: Nature of each operation	Variable cards: Columns acted on by each operation	Variable cards: Columns that receive the result of each operation	Variable cards: Indication of change of value on any column	Statement of results
${}^1V_0 = m$	1	1	×	${}^1V_0 \times {}^1V_4 =$	${}^1V_6$	${}^1V_0 = {}^1V_0$, ${}^1V_4 = {}^1V_4$	${}^1V_6 = mn'$
${}^1V_1 = n$	2	"	×	${}^1V_3 \times {}^1V_1 =$	${}^1V_7$	${}^1V_3 = {}^1V_3$, ${}^1V_1 = {}^1V_1$	${}^1V_7 = m'n$
${}^1V_2 = d$	3	"	×	${}^1V_2 \times {}^1V_4 =$	${}^1V_8$	${}^1V_2 = {}^1V_2$, ${}^1V_4 = {}^0V_4$	${}^1V_8 = dn'$
${}^1V_3 = m'$	4	"	×	${}^1V_5 \times {}^1V_1 =$	${}^1V_9$	${}^1V_5 = {}^1V_5$, ${}^1V_1 = {}^0V_1$	${}^1V_9 = d'n$
${}^1V_4 = n'$	5	"	×	${}^1V_0 \times {}^1V_5 =$	${}^1V_{10}$	${}^1V_0 = {}^0V_0$, ${}^1V_5 = {}^0V_5$	${}^1V_{10} = d'm$
${}^1V_5 = d'$	6	"	×	${}^1V_2 \times {}^1V_3 =$	${}^1V_{11}$	${}^1V_2 = {}^0V_2$, ${}^1V_3 = {}^0V_3$	${}^1V_{11} = dm'$
	7	2	−	${}^1V_6 - {}^1V_7 =$	${}^1V_{12}$	${}^1V_6 = {}^0V_6$, ${}^1V_7 = {}^0V_7$	${}^1V_{12} = mn' - m'n$
	8	"	−	${}^1V_8 - {}^1V_9 =$	${}^1V_{13}$	${}^1V_8 = {}^0V_8$, ${}^1V_9 = {}^0V_9$	${}^1V_{13} = dn' - d'n$
	9	"	−	${}^1V_{10} - {}^1V_{11} =$	${}^1V_{14}$	${}^1V_{10} = {}^0V_{10}$, ${}^1V_{11} = {}^0V_{11}$	${}^1V_{14} = d'm - dm'$
	10	3	÷	${}^1V_{13} \div {}^1V_{12} =$	${}^1V_{15}$	${}^1V_{13} = {}^0V_{13}$, ${}^1V_{12} = {}^1V_{12}$	${}^1V_{15} = \frac{dn' - d'n}{mn' - m'n} = x$
	11	"	÷	${}^1V_{14} \div {}^1V_{12} =$	${}^1V_{16}$	${}^1V_{14} = {}^0V_{14}$, ${}^1V_{12} = {}^0V_{12}$	${}^1V_{16} = \frac{d'm - dm'}{mn' - m'n} = y$
1	2	3	4	5	6	7	8

card will be changed into a subtraction through the intervention of mechanisms which are brought into play by this very difference of sign. Since the subtraction can only be effected on the larger of the two numbers, it must be arranged that the disc of signs of the larger number shall not move while the smaller of the two numbers is being effaced from its column and subtracted from the other, whence the result will have the sign of this latter, just as in fact it ought to be. The combinations to which algebraical subtraction give rise, are analogous to the preceding. Let us pass on to multiplication. When two numbers to be multiplied are of the same sign, the result is positive; if the signs are different, the product must be negative. In order that the machine may act conformably to this law, we have but to conceive that on the column containing the product of the two given numbers, the digit which indicates the sign of that product has been formed by the mutual addition of the two digits that respectively indicated the signs of the two given numbers; it is then obvious that if the digits of the signs are both even, or both odd, their sum will be an even number, and consequently will express a positive number; but that if, on the contrary, the two digits of the signs are one even and the other odd, their sum will be an odd number, and will consequently express a negative number. In the case of division. instead of adding the digits of the discs, they must be subtracted one from the other, which will produce results analogous to the preceding; that is to say, that if these figures are both even or both uneven, the remainder of this subtraction will be even; and it will be uneven in the contrary case. When I speak of mutually adding or subtracting the numbers expressed by the digits of the signs, I merely mean that one of the sign-discs is made to advance or retrograde a number of divisions equal to that which is expressed by the digit on the other sign-disc. We see, then, from the preceding explanation, that it is possible mechanically to combine the signs of quantities so as to obtain results conformable to those indicated by algebra[9].

The machine is not only capable of executing those numerical calculations which depend on a given algebraical formula, but it is also fitted for analytical calculations in which there are one or several variables to be considered. It must be assumed that the analytical expression to be operated on can be developed according to powers of the variable, or according to determinate functions of this same variable, such as circular functions, for instance; and similarly for the result that is to be attained. If we then suppose that above the columns of the store, we have inscribed the powers or the functions of the variable, arranged according to whatever is the prescribed law of development, the coefficients of these several terms may be respectively placed on the corresponding column below each. In this manner we shall have a representation of an analytical development; and, supposing the position of the several terms composing it to be invariable, the problem will be reduced to that of calculating their coefficients according to the laws demanded by the nature of the

[9] Not having had leisure to discuss with Mr. Babbage the manner of introducing into his machine the combination of algebraical signs, I do not pretend here to expose the method he uses for this purpose; but I considered that I ought myself to supply the deficiency, conceiving that this paper would have been imperfect if I had omitted to point out one means that might be employed for resolving this essential part of the problem in question.

Table IV

Columns above which are written functions of the variable	Coefficients: Given	Coefficients: To be formed	Cards of the operations: No. of the Operations	Cards of the operations: Nature of the Operation	Cards of the variables: Columns on which operations are to be performed	Columns on which are to be inscribed the results of the operations	Indication of change of value in any column submitted to an operation	Results of the operations	
x^0 ..... $^1V_0$	a	"	"	"	"	"	"	"	"
x^1 ..... $^1V_1$	b	"	"	"	"	"	"	"	"
$\cos^0 x$ ..... $^1V_2$	A	"	"	"	"	"	"	"	"
$\cos^1 x$ ..... $^1V_3$	B	"	"	"	"	"	"	"	"
$x^0 \cos^0 x$ .... $^0V_4$	...	aA	1	×	$^1V_0 \times {}^1V_2 =$	$^1V_4$	$\left\{ \begin{matrix} {}^1V_0 = {}^1V_0 \\ {}^1V_2 = {}^1V_2 \end{matrix} \right\}$	$^1V_4 = aA$	coefficients of $x^0 \cos^0 x$
$x^0 \cos^1 x$ .... $^0V_5$	...	aB	2	×	$^1V_0 \times {}^1V_3 =$	$^1V_5$	$\left\{ \begin{matrix} {}^1V_0 = {}^0V_0 \\ {}^1V_3 = {}^1V_3 \end{matrix} \right\}$	$^1V_5 = aB$	$x^0 \cos^1 x$
$x^1 \cos^0 x$ .... $^0V_6$	...	bA	3	×	$^1V_1 \times {}^1V_2 =$	$^1V_6$	$\left\{ \begin{matrix} {}^1V_1 = {}^1V_1 \\ {}^1V_2 = {}^0V_2 \end{matrix} \right\}$	$^1V_6 = bA$	$x^1 \cos^0 x$
$x^1 \cos^1 x$ .... $^0V_7$	...	bB	4	×	$^1V_1 \times {}^1V_3 =$	$^1V_7$	$\left\{ \begin{matrix} {}^1V_1 = {}^0V_1 \\ {}^1V_3 = {}^0V_3 \end{matrix} \right\}$	$^1V_7 = bB$	$x^1 \cos^1 x$

question. In order to make this more clear, we shall take the following very simple example[10], are to multiply $(a + bx^1)$ by $(A + B \cos^1 x)$. We shall begin by writing x^0 , x^1, $\cos^0 x$, $\cos^1 x$, above the columns V_0, V_1, V_2, V_3; then since, from the form of the two functions to be combined, the terms which are to compose the products will be of the following nature, $x^0 \cdot \cos^0 x$, $x^0 \cdot \cos^1 x$, $x^1 \cdot \cos^0 x$, $x^1 \cdot \cos^1 x$, these will be inscribed above the columns V_4, V_5, V_6, V_7. The coefficients of x^0, x^1, $\cos^0 x$, $\cos^1 x$ being given, they will, by means of the mill, be passed to the columns V_0, V_1, V_2 and V_3. Such are the primitive data of the problem. It is now the business of the machine to work out its solution, that is, to find the coefficients which are to be inscribed on V_4, V_5, V_6, V_7. To attain this object, the law of formation of these same coefficients being known, the machine will act through the intervention of the cards, in the manner indicated by table IV[11].

It will now be perceived that a general application may be made of the principle developed in the preceding example, to every species of process which it may be proposed to effect on series submitted to calculation. It is sufficient that the law of formation of the coefficients be known, and that this law be inscribed on the cards of the machine, which will then of itself execute all the calculations requisite for arriving at the proposed result. If, for instance, a recurring series were proposed, the law of formation of the coefficients being here uniform, the same operations which must be performed for one of them will be repeated for all the others; there will merely be a change in the locality of the operation, that is, it will be performed with different columns. Generally, since every analytical expression is susceptible of being expressed in a series ordered according to certain functions of the variable, we perceive that the machine will include all analytical calculations which can be definitively reduced to the formation of coefficients according to certain laws, and to the distribution of these with respect to the variables.

We may deduce the following important consequence from these explanations, viz. that since the cards only indicate the nature of the operations to be performed, and the columns of Variables with which they are to be executed, these cards will themselves possess all the generality of analysis, of which they are in fact merely a translation. We shall now further examine some of the difficulties which the machine must surmount, if its assimilation to analysis is to be complete. There are certain functions which necessarily change in nature when they pass through zero or infinity, or whose values cannot be admitted when they pass these limits. When such cases present themselves, the machine is able, by means of a bell, to give notice that the passage through zero or infinity is taking place, and it then stops until the attendant has again set it in action for whatever process it may next be desired that it shall perform. If this process has been foreseen, then the machine, instead of ringing, will so dispose itself as to present the new cards which have relation to the operation that is to succeed the passage through zero and infinity. These new cards may follow the first, but may only come into play contingently upon one or other of the two circumstances just mentioned taking place.

Let us consider a term of the form ab^n; since the cards are but a translation of the analytical formula, their number in this particular case must be the same, whatever be the value of n; that is to say, whatever be the number of multiplications required for elevating b to the nth power (we are supposing for the moment that n is a whole number). Now, since the exponent n indicates that b is to be multiplied n times by itself, and all these operations are of the same nature, it will be sufficient to employ one single operation-card, viz. that which orders the multiplication.

But when n is given for the particular case to be calculated, it will be further requisite that the machine limit the number of its multiplications according to the given values. The process may be thus arranged. The three numbers a, b and n will be written on as many distinct columns of the store; we shall designate them V_0, V_1, V_2; the result ab^n will place itself on the column V_3. When the number n has been introduced into the machine, a card will order a certain registering-apparatus to mark (n-1), and will at the same time execute the multiplication of b by b. When this is completed, it will be found that the registering-apparatus has effaced a unit, and that it only marks ($n-2$); while the machine will now again order the number b written on the column V_1 to multiply itself with the product b^2 written on the column V_3, which will give b^3. Another unit is then effaced from the registering-apparatus, and the same processes are continually repeated until it only marks zero. Thus the number b^n will be found inscribed on V_3, when the

[10] See Note E in section "Notes by the Translator"

[11] For an explanation of the upper left-hand indices attached to the V's in this and in the preceding Table, we must refer the reader to Note D, amongst those appended to the memoir. —NOTE BY TRANSLATOR.

machine, pursuing its course of operations, will order the product of b^n by a; and the required calculation will have been completed without there being any necessity that the number of operation-cards used should vary with the value of n. If n were negative, the cards, instead of ordering the multiplication of a by b^n, would order its division; this we can easily conceive, since every number, being inscribed with its respective sign, is consequently capable of reacting on the nature of the operations to be executed. Finally, if n were fractional, of the form p/q, an additional column would be used for the inscription of q, and the machine would bring into action two sets of processes, one for raising b to the power p, the other for extracting the qth root of the number so obtained.

Again, it may be required, for example, to multiply an expression of the form ax^m+bx^n by another Ax^p+Bx^q, and then to reduce the product to the least number of terms, if any of the indices are equal. The two factors being ordered with respect to x, the general result of the multiplication would be $Aax^{m+p}+Abx^{n+p}+Bax^{m+q}+Bbx^{n+q}$. Up to this point the process presents no difficulties; but suppose that we have $m=p$ and $n=q$, and that we wish to reduce the two middle terms to a single one $(Ab+Ba)x^{m+q}$. For this purpose, the cards may order $m+q$ and $n+p$ to be transferred into the mill, and there subtracted one from the other; if the remainder is nothing, as would be the case on the present hypothesis, the mill will order other cards to bring to it the coefficients Ab and Ba, that it may add them together and give them in this state as a coefficient for the single term $x^{n+p}=x^{m+q}$.

This example illustrates how the cards are able to reproduce all the operations which intellect performs in order to attain a determinate result, if these operations are themselves capable of being precisely defined.

Let us now examine the following expression:

$$2 \cdot \frac{2^2 \cdot 4^2 \cdot 6^2 \cdot 8^2 \cdot 10^2 \ldots (2n)^2}{1^2 \cdot 3^2 \cdot 5^2 \cdot 7^2 \cdot 9^2 \ldots (2n-1)^2 \cdot (2n+1)^2},$$

which we know becomes equal to the ratio of the circumference to the diameter, when n is infinite. We may require the machine not only to perform the calculation of this fractional expression, but further to give indication as soon as the value becomes identical with that of the ratio of the circumference to the diameter when n is infinite, a case in which the computation would be impossible. Observe that we should thus require of the machine to interpret a result not of itself evident, and that this is not amongst its attributes, since it is no thinking being. Nevertheless, when the cos of $n=1/0$ has been foreseen, a card may immediately order the substitution of the value of π (π being the ratio of the circumference to the diameter), without going through the series of calculations indicated. This would merely require that the machine contain a special card, whose office it should be to place the number π in a direct and independent manner on the column indicated to it. And here we should introduce the mention of a third species of cards, which may be called *cards of numbers*. There are certain numbers, such as those expressing the ratio of the circumference to the diameter, the Numbers of Bernoulli, &c., which frequently present themselves in calculations. To avoid the necessity for computing them every time they have to be used, certain cards may be combined specially in order to give these numbers ready made into the mill, whence they afterwards go and place themselves on those columns of the store that are destined for them. Through this means the machine will be susceptible of those simplifications afforded by the use of numerical tables. It would be equally possible to introduce, by means of these cards, the logarithms of numbers; but perhaps it might not be in this case either the shortest or the most appropriate method; for the machine might be able to perform the same calculations by other more expeditious combinations, founded on the rapidity with which it executes the first four operations of arithmetic. To give an idea of this rapidity, we need only mention that Mr. Babbage believes he can, by his engine, form the product of two numbers, each containing twenty figures, in *three minutes*.

Perhaps the immense number of cards required for the solution of any rather complicated problem may appear to be an obstacle; but this does not seem to be the case. There is no limit to the number of cards that can be used. Certain stuffs require for their fabrication not less than *twenty thousand* cards, and we may unquestionably far exceed even this quantity[12].

Resuming what we have explained concerning the Analytical Engine, we may conclude that it is based on two principles: the first consisting in the fact that every arithmetical calculation ultimately depends on four principal operations—addition, subtraction, multiplication, and division; the second, in the possibility of reducing every analytical calculation to that of the coefficients for the several terms of a series. If this last principle be true, all the operations of analysis come within the domain of the engine. To take another point of view: the use of the cards offers a generality equal to that of algebraical formulæ, since such a formula simply indicates the nature and order of the operations requisite for arriving at a certain definite result, and similarly the cards merely command the engine to perform these same operations; but in order that the mechanisms may be able to act to any purpose, the numerical data of the problem must in every particular case be introduced. Thus the same series of cards will serve for all questions whose sameness of nature is such as to require nothing altered excepting the numerical data. In this light the cards are merely a translation of algebraical formulæ, or, to express it better, another form of analytical notation.

Since the engine has a mode of acting peculiar to itself, it will in every particular case be necessary to arrange the series of calculations conformably to the means which the machine possesses; for such or such a process which might be very easy for a calculator may be long and complicated for the engine, and *vice versâ*.

Considered under the most general point of view, the essential object of the machine being to calculate, according to the laws dictated to it, the values of numerical coefficients which it is then to distribute appropriately on the columns which represent the variables, it follows that the interpretation of formulæ and of results is beyond its province, unless indeed this very interpretation be itself susceptible of expression by means of the symbols which the machine employs. Thus, although it is not itself the being that reflects, it may yet be considered as the being

[12] See Note F in section "Notes by the Translator"

which executes the conceptions of intelligence[13]. The cards receive the impress of these conceptions, and transmit to the various trains of mechanism composing the engine the orders necessary for their action. When once the engine shall have been constructed, the difficulty will be reduced to the making out of the cards; but as these are merely the translation of algebraical formulæ, it will, by means of some simple notations, be easy to consign the execution of them to a workman. Thus the whole intellectual labour will be limited to the preparation of the formulæ, which must be adapted for calculation by the engine.

Now, admitting that such an engine can be constructed, it may be inquired: what will be its utility? To recapitulate; it will afford the following advantages:—First, rigid accuracy. We know that numerical calculations are generally the stumbling-block to the solution of problems, since errors easily creep into them, and it is by no means always easy to detect these errors. Now the engine, by the very nature of its mode of acting, which requires no human intervention during the course of its operations, presents every species of security under the head of correctness: besides, it carries with it its own check; for at the end of every operation it prints off, not only the results, but likewise the numerical data of the question; so that it is easy to verify whether the question has been correctly proposed. Secondly, economy of time: to convince ourselves of this, we need only recollect that the multiplication of two numbers, consisting each of twenty figures, requires at the very utmost three minutes. Likewise, when a long series of identical computations is to be performed, such as those required for the formation of numerical tables, the machine can be brought into play so as to give several results at the same time, which will greatly abridge the whole amount of the processes. Thirdly, economy of intelligence: a simple arithmetical computation requires to be performed by a person possessing some capacity; and when we pass to more complicated calculations, and wish to use algebraical formulæ in particular cases, knowledge must be possessed which presupposes preliminary mathematical studies of some extent. Now the engine, from its capability of performing by itself all these purely material operations, spares intellectual labour, which may be more profitably employed. Thus the engine may be considered as a real manufactory of figures, which will lend its aid to those many useful sciences and arts that depend on numbers. Again, who can foresee the consequences of such an invention? In truth, how many precious observations remain practically barren for the progress of the sciences, because there are not powers sufficient for computing the results! And what discouragement does the perspective of a long and arid computation cast into the mind of a man of genius, who demands time exclusively for meditation, and who beholds it snatched from him by the material routine of operations! Yet it is by the laborious route of analysis that he must reach truth; but he cannot pursue this unless guided by numbers; for without numbers it is not given us to raise the veil which envelopes the mysteries of nature. Thus the idea of constructing an apparatus capable of aiding human weakness in such researches, is a conception which, being realized, would mark a glorious epoch in the history of the sciences. The plans have been arranged for all the various parts, and for all the wheel-work, which compose this immense apparatus, and their action studied; but these have not yet been fully combined together in the drawings[14] and mechanical notation[15]. The confidence which the genius of Mr. Babbage must inspire, affords legitimate ground for hope that this enterprise will be crowned with success; and while we render homage to the intelligence which directs it, let us breathe aspirations for the accomplishment of such an undertaking.

[13] See Note G in section "Notes by the Translator"

NOTES BY THE TRANSLATOR

Note A

The particular function whose integral the Difference Engine was constructed to tabulate, is

$$\Delta^7 u_x = 0.$$

The purpose which that engine has been specially intended and adapted to fulfil, is the computation of nautical and astronomical tables. The integral of

$$\Delta^7 u_x = 0$$

being $u_z = a+bx+cx^2+dx^3+ex^4+fx^5+gx^6$,

the constants *a*, *b*, *c*, &c. are represented on the seven columns of discs, of which the engine consists. It can therefore tabulate *accurately* and to an *unlimited extent*, all series whose general term is comprised in the above formula; and it can also tabulate *approximatively* between *intervals of greater or less extent*, all other series which are capable of tabulation by the Method of Differences.

The Analytical Engine, on the contrary, is not merely adapted for *tabulating* the results of one particular function and of no other, but for *developing and tabulating* any function whatever. In fact the engine may be described as being the material expression of any indefinite function of any degree of generality and complexity, such as for instance,

$$F(x, y, z, \log x, \sin y, x^p, \&c.),$$

which is, it will be observed, a function of all other possible functions of any number of quantities.

In this, which we may call the *neutral* or *zero* state of the engine, it is ready to receive at any moment, by means of cards constituting a portion of its mechanism (and applied on the principle of those used in the Jacquard-loom), the impress of whatever *special* function we may desire to develope or to tabulate. These cards contain within themselves (in a manner explained in the Memoir itself) the law of development of the particular function that may be under consideration, and they compel the mechanism to act accordingly in a certain corresponding order. One of the simplest cases would be for example, to suppose that

[14] This sentence has been slightly altered in the translation in order to express more exactly the present state of the engine. —NOTE BY TRANSLATOR.

[15] The notation here alluded to is a most interesting and important subject, and would have well deserved a separate and detailed Note upon it amongst those appended to the Memoir. It has, however, been impossible, within the space allotted, even to touch upon so wide a field. —NOTE BY TRANSLATOR.

$$F(x, y, z, \&c.\ \&c.)$$

is the particular function

$$\Delta^n u_z = 0$$

which the Difference Engine tabulates for values of *n* only up to 7. In this case the cards would order the mechanism to go through that succession of operations which would tabulate

$$u_z = a + bx + cx^2 + \cdots + mx^{n-1}$$

where *n* might be any number whatever.

These cards, however, have nothing to do with the regulation of the particular *numerical* data. They merely determine the operations[16] to be effected, which operations may of course be performed on an infinite variety of particular numerical values, and do not bring out any definite numerical results unless the numerical data of the problem have been impressed on the requisite portions of the train of mechanism. In the above example, the first essential step towards an arithmetical result would be the substitution of specific numbers for *n*, and for the other primitive quantities which enter into the function.

Again, let us suppose that for F we put two complete equations of the fourth degree between x and y. We must then express on the cards the law of elimination for such equations. The engine would follow out those laws, and would ultimately give the equation of one variable which results from such elimination. Various modes of elimination might be selected; and of course the cards must be made out accordingly. The following is one mode that might be adopted. The engine is able to multiply together any two functions of the form

$$a + bx + cx^2 + \cdots + px^n.$$

This granted, the two equations may be arranged according to the powers of *y*, and the coefficients of the powers of *y* may be arranged according to powers of *x*. The elimination of *y* will result from the successive multiplications and subtractions of several such functions. In this, and in all other instances, as was explained above, the particular *numerical* data and the *numerical* results are determined by means and by portions of the mechanism which act quite independently of those that regulate the *operations*.

In studying the action of the Analytical Engine, we find that the peculiar and independent nature of the considerations which in all mathematical analysis belong to *operations*, as distinguished from *the objects operated upon* and from the *results* of the operations performed upon those objects, is very strikingly defined and separated.

It is well to draw attention to this point, not only because its full appreciation is essential to the attainment of any very just and adequate general comprehension of the powers and mode of action of the Analytical Engine, but also because it is one which is perhaps too little kept in view in the study of mathematical science in general. It is, however, impossible to confound it with other considerations, either when we trace the manner in which that engine attains its results, or when we prepare the data for its attainment of those results. It were much to be desired, that when mathematical processes pass through the human brain instead of through the medium of inanimate mechanism, it were equally a necessity of things that the reasonings connected with operations should hold the same just place as a clear and well-defined branch of the subject of analysis, a fundamental but yet independent ingredient in the science, which they must do in studying the engine. The confusion, the difficulties, the contradictions which, in consequence of a want of accurate distinctions in this particular, have up to even a recent period encumbered mathematics in all those branches involving the consideration of negative and impossible quantities, will at once occur to the reader who is at all versed in this science, and would alone suffice to justify dwelling somewhat on the point, in connexion with any subject so peculiarly fitted to give forcible illustration of it as the Analytical Engine. It may be desirable to explain, that by the word operation, we mean any process which alters the mutual relation of two or more things, be this relation of what kind it may. This is the most general definition, and would include all subjects in the universe. In abstract mathematics, of course operations alter those particular relations which are involved in the considerations of number and space, and the results of operations are those peculiar results which correspond to the nature of the subjects of operation. But the science of operations, as derived from mathematics more especially, is a science of itself, and has its own abstract truth and value; just as logic has its own peculiar truth and value, independently of the subjects to which we may apply its reasonings and processes. Those who are accustomed to some of the more modern views of the above subject, will know that a few fundamental relations being true, certain other combinations of relations must of necessity follow; combinations unlimited in variety and extent if the deductions from the primary relations be carried on far enough. They will also be aware that one main reason why the separate nature of the science of operations has been little felt, and in general little dwelt on, is the shifting meaning of many of the symbols used in mathematical notation. First, the symbols of operation are frequently also the symbols of the results of operations. We may say that these symbols are apt to have both a retrospective and a prospective signification. They may signify either relations that are the consequences of a series of processes already performed, or relations that are yet to be effected through certain processes. Secondly, figures, the symbols of numerical magnitude, are frequently also the symbols of *operations*, as when they are the indices of powers. Wherever terms have a shifting meaning, independent sets of considerations are liable to become complicated together, and reasonings and results are frequently falsified. Now in the Analytical Engine, the operations which come under the first of the above heads are ordered and combined by means of a notation and of a train of mechanism which belong exclusively to themselves; and with respect to the second head, whenever numbers meaning *operations* and not *quantities* (such as the indices of powers) are inscribed on any column or set of columns, those columns immediately act in a wholly separate and independent manner, becoming connected with the *operating mechanism* exclusively, and re-acting upon this. They never come into combination with

[16] We do not mean to imply that the *only* use made of the Jacquard cards is that of regulating the algebraical *operations*; but we mean to explain that *those* cards and portions of mechanism which regulate these *operations* are wholly independent of those which are used for other purposes. M. Menabrea explains that there are *three* classes of cards used in the engine for three distinct sets of objects, viz. *Cards of the Operations*, *Cards of the Variables*, and certain *Cards of Numbers*.

numbers upon any other columns meaning *quantities*; though, of course, if there are numbers meaning *operations* upon *n* columns, these may *combine amongst each other*, and will often be required to do so, just as numbers meaning *quantities* combine with each other in any variety. It might have been arranged that all numbers meaning *operations* should have appeared on some separate portion of the engine from that which presents numerical *quantities*; but the present mode is in some cases more simple, and offers in reality quite as much distinctness when understood.

The operating mechanism can even be thrown into action independently of any object to operate upon (although of course no *result* could then be developed). Again, it might act upon other things besides *number*, were objects found whose mutual fundamental relations could be expressed by those of the abstract science of operations, and which should be also susceptible of adaptations to the action of the operating notation and mechanism of the engine. Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent.

The Analytical Engine is an *embodying of the science of operations*, constructed with peculiar reference to abstract number as the subject of those operations. The Difference Engine is the embodying of *one particular and very limited set of operations*, which (see the notation used in Note B) may be expressed thus (+, +, +, +, +, +), or thus, 6(+). Six repetitions of the one operation, +, is, in fact, the whole sum and object of that engine. It has seven columns, and a number on any column can add itself to a number on the next column to its *right-hand*. So that, beginning with the column furthest to the left, six additions can be effected, and the result appears on the seventh column, which is the last on the right-hand. The *operating* mechanism of this engine acts in as separate and independent a manner as that of the Analytical Engine; but being susceptible of only one unvarying and restricted combination, it has little force or interest in illustration of the distinct nature of the *science of operations*. The importance of regarding the Analytical Engine under this point of view will, we think, become more and more obvious as the reader proceeds with M. Menabrea's clear and masterly article. The calculus of operations is likewise in itself a topic of so much interest, and has of late years been so much more written on and thought on than formerly, that any bearing which that engine, from its mode of constitution, may possess upon the illustration of this branch of mathematical science should not be overlooked. Whether the inventor of this engine had any such views in his mind while working out the invention, or whether he may subsequently ever have regarded it under this phase, we do not know; but it is one that forcibly occurred to ourselves on becoming acquainted with the means through which analytical combinations are actually attained by the mechanism. We cannot forbear suggesting one practical result which it appears to us must be greatly facilitated by the independent manner in which the engine orders and combines its *operations*: we allude to the attainment of those combinations into which *imaginary quantities* enter. This is a branch of its processes into which we have not had the opportunity of inquiring, and our conjecture therefore as to the principle on which we conceive the accomplishment of such results may have been made to depend, is very probably not in accordance with the fact, and less subservient for the purpose than some other principles, or at least requiring the cooperation of others. It seems to us obvious, however, that where operations are so independent in their mode of acting, it must be easy, by means of a few simple provisions, and additions in arranging the mechanism, to bring out a *double* set of *results*, viz.—1st, the *numerical magnitudes* which are the results of operations performed on *numerical data*. (These results are the *primary* object of the engine.) 2ndly, the *symbolical results* to be attached to those numerical results, which symbolical results are not less the necessary and logical consequences of operations performed upon *symbolical data*, than are numerical results when the data are numerical[17].

If we compare together the powers and the principles of construction of the Difference and of the Analytical Engines, we shall perceive that the capabilities of the latter are immeasurably more extensive than those of the former, and that they in fact hold to each other the same relationship as that of analysis to arithmetic. The Difference Engine can effect but one particular series of operations, viz. that required for tabulating the integral of the special function

$$\Delta^n u_z = 0;$$

and as it can only do this for values of *n* up to 7[18], it cannot be considered as being the most *general* expression even of *one particular* function, much less as being the expression of any and all possible functions of all degrees of generality. The Difference Engine can in reality (as has been already partly explained) do nothing but *add*; and any other processes, not excepting those of simple subtraction, multiplication and division, can be performed by it only just to that extent in which it is possible, by judicious mathematical arrangement and artifices, to reduce them to a *series of additions*. The method of differences is, in fact, a method of additions; and as it includes within its means a larger number of results attainable by *addition* simply, than any other mathematical principle, it was very appropriately selected as the basis on which to construct *an Adding Machine*, so as to give to the powers of such a machine the widest possible range. The Analytical Engine, on the contrary, can either add, subtract, multiply or divide with equal facility; and performs each of these four operations in a direct manner, without the aid of any of the other three. This one

[17] In fact, such an extension as we allude to would merely constitute a further and more perfected development of any system introduced for making the proper combinations of the signs *plus* and *minus*. How ably M. Menabrea has touched on this restricted case is pointed out in Note B.

[18] The machine might have been constructed so as to tabulate for a higher value of n than seven. Since, however, every unit added to the value of n increases the extent of the mechanism requisite, there would on this account be a limit beyond which it could not be practically carried. Seven is sufficiently high for the calculation of all ordinary tables.

The fact that, in the Analytical Engine, the same extent of mechanism suffices for the solution of $\Delta^n u_z = 0$, hether n=7, n=100,000, or n=any number whatever, at once suggests how entirely distinct must be the nature of the principles through whose application matter has been enabled to become the working agent of abstract mental operations in each of these engines respectively, and it affords an equally obvious presumption, that in the case of the Analytical Engine, not only are those principles in themselves of a higher and more comprehensive description, but also such as must vastly extend the practical value of the engine whose basis they constitute.

fact implies everything; and it is scarcely necessary to point out, for instance, that while the Difference Engine can merely *tabulate*, and is incapable of *developing*, the Analytical Engine can either *tabulate or develope*.

The former engine is in its nature strictly *arithmetical*, and the results it can arrive at lie within a very clearly defined and restricted range, while there is no finite line of demarcation which limits the powers of the Analytical Engine. These powers are co-extensive with our knowledge of the laws of analysis itself, and need be bounded only by our acquaintance with the latter. Indeed we may consider the engine as the *material and mechanical representative* of analysis, and that our actual working powers in this department of human study will be enabled more effectually than heretofore to keep pace with our theoretical knowledge of its principles and laws, through the complete control which the engine gives us over the *executive manipulation* of algebraical and numerical symbols.

Those who view mathematical science, not merely as a vast body of abstract and immutable truths, whose intrinsic beauty, symmetry and logical completeness, when regarded in their connexion together as a whole, entitle them to a prominent place in the interest of all profound and logical minds, but as possessing a yet deeper interest for the human race, when it is remembered that this science constitutes the language through which alone we can adequately express the great facts of the natural world, and those unceasing changes of mutual relationship which, visibly or invisibly, consciously or unconsciously to our immediate physical perceptions, are interminably going on in the agencies of the creation we live amidst: those who thus think on mathematical truth as the instrument through which the weak mind of man can most effectually read his Creator's works, will regard with especial interest all that can tend to facilitate the translation of its principles into explicit practical forms.

The distinctive characteristic of the Analytical Engine, and that which has rendered it possible to endow mechanism with such extensive faculties as bid fair to make this engine the executive right-hand of abstract algebra, is the introduction into it of the principle which Jacquard devised for regulating, by means of punched cards, the most complicated patterns in the fabrication of brocaded stuffs. It is in this that the distinction between the two engines lies. Nothing of the sort exists in the Difference Engine. We may say most aptly, that the Analytical Engine *weaves algebraical patterns* just as the Jacquard-loom weaves flowers and leaves. Here, it seems to us, resides much more of originality than the Difference Engine can be fairly entitled to claim. We do not wish to deny to this latter all such claims. We believe that it is the only proposal or attempt ever made to construct a calculating machine *founded on the principle of successive orders of differences*, and capable of *printing off its own results*; and that this engine surpasses its predecessors, both in the extent of the calculations which it can perform, in the facility, certainty and accuracy with which it can effect them, and in the absence of all necessity for the intervention of human intelligence *during the performance of its calculations*. Its nature is, however, limited to the strictly arithmetical, and it is far from being the first or only scheme for constructing *arithmetical* calculating machines with more or less of success.

The bounds of *arithmetic* were however outstepped the moment the idea of applying the cards had occurred; and the Analytical Engine does not occupy common ground with mere “calculating machines.” It holds a position wholly its own; and the considerations it suggests are most interesting in their nature. In enabling mechanism to combine together *general* symbols in successions of unlimited variety and extent, a uniting link is established between the operations of matter and the abstract mental processes of the *most abstract* branch of mathematical science. A new, a vast, and a powerful language is developed for the future use of analysis, in which to wield its truths so that these may become of more speedy and accurate practical application for the purposes of mankind than the means hitherto in our possession have rendered possible. Thus not only the mental and the material, but the theoretical and the practical in the mathematical world, are brought into more intimate and effective connexion with each other. We are not aware of its being on record that anything partaking in the nature of what is so well designated the *Analytical* Engine has been hitherto proposed, or even thought of, as a practical possibility, any more than the idea of a thinking or of a reasoning machine.

We will touch on another point which constitutes an important distinction in the modes of operating of the Difference and Analytical Engines. In order to enable the former to do its business, it is necessary to put into its columns the series of numbers constituting the first terms of the several orders of differences for whatever is the particular table under consideration. The machine then works *upon* these as its data. But these data must themselves have been already computed through a series of calculations by a human head. Therefore that engine can only produce results depending on data which have been arrived at by the explicit and actual working out of processes that are in their nature different from any that come within the sphere of its own powers. In other words, an *analysing* process must have been gone through by a human mind in order to obtain the data upon which the engine then *synthetically* builds its results. The Difference Engine is in its character exclusively *synthetical*, while the Analytical Engine is equally capable of analysis or of synthesis.

It is true that the Difference Engine can calculate to a much greater extent with these few preliminary data, than the data themselves required for their own determination. The table of squares, for instance, can be calculated to any extent whatever, when the numbers *one* and *two* are furnished; and a very few differences computed at any part of a table of logarithms would enable the engine to calculate many hundreds or even thousands of logarithms. Still the circumstance of its requiring, as a previous condition, that any function whatever shall have been numerically worked out, makes it very inferior in its nature and advantages to an engine which, like the Analytical Engine, requires merely that we should know the *succession and distribution* of the operations to be performed; without there being any occasion[19], in order to obtain data on which it can work, for our ever having gone through either the same particular operations which it is itself to effect, or any others. Numerical data must of course be given it,

[19] This subject is further noticed in Note F.

but they are mere arbitrary ones; not data that could only be arrived at through a systematic and necessary series of previous numerical calculations, which is quite a different thing.

To this it may be replied, that an analysing process must equally have been performed in order to furnish the Analytical Engine with the necessary *operative* data; and that herein may also lie a possible source of error. Granted that the actual mechanism is unerring in its processes, the *cards* may give it wrong orders. This is unquestionably the case; but there is much less chance of error, and likewise far less expenditure of time and labour, where operations only, and the distribution of these operations, have to be made out, than where explicit numerical results are to be attained. In the case of the Analytical Engine we have undoubtedly to lay out a certain capital of analytical labour in one particular line; but this is in order that the engine may bring us in a much larger return in another line. It should be remembered also that the cards, when once made out for any formula, have all the generality of algebra, and include an infinite number of particular cases.

We have dwelt considerably on the distinctive peculiarities of each of these engines, because we think it essential to place their respective attributes in strong relief before the apprehension of the public; and to define with clearness and accuracy the wholly different nature of the principles on which each is based, so as to make it self-evident to the reader (the mathematical reader at least) in what manner and degree the powers of the Analytical Engine transcend those of an engine, which, like the Difference Engine, can only work out such results as may be derived from *one restricted and particular series of processes*, such as those included in $\Delta^n u_z = 0$. We think this of importance, because we know that there exists considerable vagueness and inaccuracy in the mind of persons in general on the subject. There is a misty notion amongst most of those who have attended at all to it, that *two* "calculating machines" have been successively invented by the same person within the last few years; while others again have never heard but of the one original "calculating machine," and are not aware of there being any extension upon this. For either of these two classes of persons the above considerations are appropriate. While the latter require a knowledge of the fact that there *are two* such inventions, the former are not less in want of accurate and well-defined information on the subject. No very clear or correct ideas prevail as to the characteristics of each engine, or their respective advantages or disadvantages; and in meeting with those incidental allusions, of a more or less direct kind, which occur in so many publications of the day, to these machines, it must frequently be matter of doubt *which* "calculating machine" is referred to, or whether *both* are included in the general allusion.

We are desirous likewise of removing two misapprehensions which we know obtain, to some extent, respecting these engines. In the first place it is very generally supposed that the Difference Engine, after it had been completed up to a certain point, *suggested* the idea of the Analytical Engine; and that the second is in fact the improved offspring of the first, and *grew out* of the existence of its predecessor, through some natural or else accidental combination of ideas suggested by this one. Such a supposition is in this instance contrary to the facts; although it seems to be almost an obvious inference, wherever two inventions, similar in their nature and objects, succeed each other closely in order of *time*, and strikingly in order of *value*; more especially when the same individual is the author of both. Nevertheless the ideas which led to the Analytical Engine occurred in a manner wholly independent of any that were connected with the Difference Engine. These ideas are indeed in their own intrinsic nature independent of the latter engine, and might equally have occurred had it never existed nor been even thought of at all.

The second of the misapprehensions above alluded to relates to the well-known suspension, during some years past, of all progress in the construction of the Difference Engine. Respecting the circumstances which have interfered with the actual completion of either invention, we offer no opinion; and in fact are not possessed of the data for doing so, had we the inclination. But we know that some persons suppose these obstacles (be they what they may) to have arisen *in consequence* of the subsequent invention of the Analytical Engine while the former was in progress. We have ourselves heard it even *lamented* that an idea should ever have occurred at all, which had turned out to be merely the means of arresting what was already in a course of successful execution, without substituting the superior invention in its stead. This notion we can contradict in the most unqualified manner. The progress of the Difference Engine had long been suspended, before there were even the least crude glimmerings of any invention superior to it. Such glimmerings, therefore, and their subsequent development, were in no way the original *cause* of that suspension; although, where difficulties of some kind or other evidently already existed, it was not perhaps calculated to remove or lessen them that an invention should have been meanwhile thought of, which, while including all that the first was capable of, possesses powers so extended as to eclipse it altogether.

We leave it for the decision of each individual (*after he has possessed himself* of competent information as to the characteristics of each engine) to determine how far it ought to be matter of regret that such an accession has been made to the powers of human science, even if it *has* (which we greatly doubt) increased to a certain limited extent some already existing difficulties that had arisen in the way of completing a valuable but lesser work. We leave it for each to satisfy himself as to the wisdom of desiring the obliteration (were that now possible) of all records of the more perfect invention, in order that the comparatively limited one might be finished. The Difference Engine would doubtless fulfil all those practical objects which it was originally destined for. It would certainly calculate all the tables that are more directly necessary for the physical purposes of life, such as nautical and other computations. Those who incline to very strictly utilitarian views may perhaps feel that the peculiar powers of the Analytical Engine bear upon questions of abstract and speculative science, rather than upon those involving every-day and ordinary human interests. These persons being likely to possess but little sympathy, or possibly acquaintance, with any branches of science which they do not find to be *useful* (according to *their* definition of that word), may conceive that the undertaking of that engine, now that the other one is already in progress, would be a barren and unproductive laying out of yet more money and labour; in fact, a work of supererogation. Even in the utilitarian aspect, however, we do not doubt that very

valuable practical results would be developed by the extended faculties of the Analytical Engine; some of which results we think we could now hint at, had we the space; and others, which it may not yet be possible to foresee, but which would be brought forth by the daily increasing requirements of science, and by a more intimate practical acquaintance with the powers of the engine, were it in actual existence.

On general grounds, both of an *a priori* description as well as those founded on the scientific history and experience of mankind, we see strong presumptions that such would be the case. Nevertheless all will probably concur in feeling that the completion of the Difference Engine would be far preferable to the non-completion of any calculating engine at all. With whomsoever or wheresoever may rest the present causes of difficulty that apparently exist towards either the completion of the old engine, or the commencement of the new one, we trust they will not ultimately result in this generation's being acquainted with these inventions through the medium of pen, ink and paper merely; and still more do we hope, that for the honour of our country's reputation in the future pages of history, these causes will not lead to the completion of the undertaking by some *other* nation or government. This could not but be matter of just regret; and equally so, whether the obstacles may have originated in private interests and feelings, in considerations of a more public description, or in causes combining the nature of both such solutions.

We refer the reader to the 'Edinburgh Review' of July 1834, for a very able account of the Difference Engine. The writer of the article we allude to has selected as his prominent matter for exposition, a wholly different view of the subject from that which M. Menabrea has chosen. The former chiefly treats it under its mechanical aspect, entering but slightly into the mathematical principles of which that engine is the representative, but giving, in considerable length, many details of the mechanism and contrivances by means of which it tabulates the various orders of differences. M. Menabrea, on the contrary, exclusively developes the analytical view; taking it for granted that mechanism is able to perform certain processes, but without attempting to explain *how*; and devoting his whole attention to explanations and illustrations of the manner in which analytical laws can be so arranged and combined as to bring every branch of that vast subject within the grasp of the assumed powers of mechanism. It is obvious that, in the invention of a calculating engine, these two branches of the subject are equally essential fields of investigation, and that on their mutual adjustment, one to the other, must depend all success. They must be made to meet each other, so that the weak points in the powers of either department may be compensated by the strong points in those of the other. They are indissolubly connected, though so different in their intrinsic nature, that perhaps the same mind might not be likely to prove equally profound or successful in both. We know those who doubt whether the powers of mechanism will in practice prove adequate in all respects to the demands made upon them in the working of such complicated trains of machinery as those of the above engines, and who apprehend that unforeseen practical difficulties and disturbances will arise in the way of accuracy and of facility of operation. The Difference Engine, however, appears to us to be in a great measure an answer to these doubts. It is complete as far as it goes, and it does work with all the anticipated success. The Analytical Engine, far from being more complicated, will in many respects be of simpler construction; and it is a remarkable circumstance attending it, that with very *simplified* means it is so much more powerful.

The article in the 'Edinburgh Review' was written some time previous to the occurrence of any ideas such as afterwards led to the invention of the Analytical Engine; and in the nature of the Difference Engine there is much less that would invite a writer to take exclusively, or even prominently, the mathematical view of it, than in that of the Analytical Engine; although mechanism has undoubtedly gone much further to meet mathematics, in the case of this engine, than of the former one. Some publication embracing the *mechanical* view of the Analytical Engine is a desideratum which we trust will be supplied before long.

Those who may have the patience to study a moderate quantity of rather dry details will find ample compensation, after perusing the article of 1834, in the clearness with which a succinct view will have been attained of the various practical steps through which mechanism can accomplish certain processes; and they will also find themselves still further capable of appreciating M. Menabrea's more comprehensive and generalized memoir. The very difference in the style and object of these two articles makes them peculiarly valuable to each other; at least for the purposes of those who really desire something more than a merely superficial and popular comprehension of the subject of calculating engines.

A. A. L.

Note B

That portion of the Analytical Engine here alluded to is called the storehouse. It contains an indefinite number of the columns of discs described by M. Menabrea. The reader may picture to himself a pile of rather large draughtsmen heaped perpendicularly one above another to a considerable height, each counter having the digits from 0 to 9 inscribed on its *edge* at equal intervals; and if he then conceives that the counters do not actually lie one upon another so as to be in contact, but are fixed at small intervals of vertical distance on a common axis which passes perpendicularly through their centres, and around which each disc can *revolve horizontally* so that any required digit amongst those inscribed on its margin can be brought into view, he will have a good idea of one of these columns. The *lowest* of the discs on any column belongs to the units, the next above to the tens, the next above this to the hundreds, and so on. Thus, if we wished to inscribe 1345 on a column of the engine, it would stand thus:

1
3
4
5

In the Difference Engine there are seven of these columns placed side by side in a row, and the working mechanism extends behind them: the general form of the whole mass of machinery is that of a quadrangular prism (more or less approaching to the cube); the results always appearing on that perpendicular face of the engine which contains the columns of discs, opposite to which face a spectator may place himself. In the Analytical Engine there would be many more of these columns, probably at least two hundred. The precise form and arrangement which the whole mass of its mechanism will assume is not yet finally determined.

We may conveniently represent the columns of discs on paper in a diagram like the following:

V_1	V_2	V_3	V_4	&c.
◯	◯	◯	◯	&c.
0	0	0	0	
0	0	0	0	
0	0	0	0	&c.
0	0	0	0	
□	□	□	□	&c.

The V's are for the purpose of convenient reference to any column, either in writing or speaking, and are consequently numbered. The reason why the letter V is chosen for the purpose in preference to any other letter, is because these columns are designated (as the reader will find in proceeding with the Memoir) the *Variables*, and sometimes the *Variable columns*, or the *columns of Variables*. The origin of this appellation is, that the values on the columns are destined to change, that is to *vary*, in every conceivable manner. But it is necessary to guard against the natural misapprehension that the columns are only intended to receive the values of the *variables* in an analytical formula, and not of the *constants*. The columns are called Variables on a ground wholly unconnected with the *analytical* distinction between constants and variables. In order to prevent the possibility of confusion, we have, both in the translation and in the notes, written Variable with a capital letter when we use the word to signify a *column of the engine*, and variable with a small letter when we mean the *variable of a formula*. Similarly, *Variable-cards* signify any cards that belong to a column of the engine.

To return to the explanation of the diagram: each circle at the top is intended to contain the algebraic sign + or −, either of which can be substituted[20] for the other, according as the number represented on the column below is positive or negative. In a similar manner any other purely *symbolical* results of algebraical processes might be made to appear in these circles. In Note A. the practicability of developing *symbolical* with no less ease than *numerical* results has been touched on. The zeros beneath the *symbolic* circles represent each of them a disc, supposed to have the digit 0 presented in front. Only four tiers of zeros have been figured in the diagram, but these may be considered as representing thirty or forty, or any number of tiers of discs that may be required. Since each disc can present any digit, and each circle any sign, the discs of every column may be so adjusted[21] as to express any positive or negative number whatever within the limits of the machine; which limits depend on the *perpendicular* extent of the mechanism, that is, on the number of discs to a column.

Each of the squares below the zeros is intended for the inscription of any *general* symbol or combination of symbols we please; it being understood that the number represented on the column immediately above is the numerical value of that symbol, or combination of symbols. Let us, for instance, represent the three quantities *a*, *n*, *x*, and let us further suppose that $a = 5$, $n = 7$, $x = 98$. We should have[22]

V_1	V_2	V_3	V_4	&c.
+†	+	+	+	
0	0	0	0	
0	0	0	0	&c.
0	0	9	0	
5	7	8	0	&c.
a	n	x		

We may now combine these symbols in a variety of ways, so as to form any required function or functions of them, and we may then inscribe each such function below brackets, every bracket uniting together those quantities (and those only) which enter into the function inscribed below it. We must also, when we have decided on the particular function whose numerical value we desire to calculate, assign another column to the right-hand for receiving the *results*, and must inscribe the function in the square below this column. In the above instance we might have any one of the following functions:

$$ax^n,\ x^{an},\ a \cdot n \cdot x,\ \tfrac{a}{n}x,\ a + n + x,\ \&c.\ \&.c$$

Let us select the first. It would stand as follows, previous to calculation:

V_1	V_2	V_3	V_4	&c.
+	+	+	+	
0	0	0	0	&c.
0	0	0	0	
0	0	9	0	
5	7	8	0	&c.
a	n	x	ax^n	
		$\underbrace{\quad\quad}_{ax^n}$		

The data being given, we must now put into the engine the cards proper for directing the operations in the case of the particular function chosen. These operations would in this instance be,

First, six multiplications in order to get x^n ($=98^7$ for the above particular data).

Secondly, one multiplication in order then to get $a \cdot x^n$ ($=5 \cdot 98^7$).

In all, seven multiplications to complete the whole process. We may thus represent them:

(×, ×, ×, ×, ×, ×, ×), or 7 (×).

The multiplications would, however, at successive stages in the solution of the problem, operate on pairs of numbers, derived from *different* columns. In other words, the *same operation* would

[20] A fuller account of the manner in which the signs are regulated is given in M. Menabrea's Memoir. He himself expresses doubts (in a note of his own) as to his having been likely to hit on the precise methods really adopted; his explanation being merely a conjectural one. That it *does* accord precisely with the fact is a remarkable circumstance, and affords a convincing proof how completely M. Menabrea has been imbued with the true spirit of the invention. Indeed the whole of the above Memoir is a striking production, when we consider that M. Menabrea had had but very slight means for obtaining any adequate ideas respecting the Analytical Engine. It requires however a considerable acquaintance with the abstruse and complicated nature of such a subject, in order fully to appreciate the penetration of the writer who could take so just and comprehensive a view of it upon such limited opportunity.

[21] This adjustment is done by hand merely.

[22] It is convenient to omit the circles whenever the signs + or − can be actually represented.

be performed on different *subjects of operation*. And here again is an illustration of the remarks made in the preceding Note[23] on the independent manner in which the engine directs its *operations*. In determining the value of ax^n, the *operations* are *homogeneous*, but are distributed amongst different *subjects of operation*, at successive stages of the computation. It is by means of certain punched cards, belonging to the Variables themselves, that the action of the operations is so *distributed* as to suit each particular function. The *Operation-cards* merely determine the succession of operations in a general manner. They in fact throw all that portion of the mechanism included in the *mill* into a series of different *states*, which we may call the *adding state*, or the *multiplying state*, &c. respectively. In each of these states the mechanism is ready to act in the way peculiar to that state, on any pair of numbers which may be permitted to come within its sphere of action. Only *one* of these operating states of the mill can exist at a time; and the nature of the mechanism is also such that only *one pair of numbers* can be received and acted on at a time. Now, in order to secure that the mill shall receive a constant supply of the proper pairs of numbers in succession, and that it shall also rightly locate the result of an operation performed upon any pair, each Variable has cards of its own belonging to it. It has, first, a class of cards whose business it is to *allow* the number on the Variable to pass into the mill, there to be operated upon. These cards may be called the *Supplying-cards*. They furnish the mill with its proper food. Each Variable has, secondly, another class of cards, whose office it is to allow the Variable to *receive* a number *from* the mill. These cards may be called the *Receiving-cards*. *They* regulate the location of results, whether temporary or ultimate results. The Variable-cards in general (including both the preceding classes) might, it appears to us, be even more appropriately designated the Distributive-cards, since it is through their means that the action of the operations, and the results of this action, are rightly *distributed*.

There are *two varieties* of the *Supplying* Variable-cards, respectively adapted for fulfilling two distinct subsidiary purposes: but as these modifications do not bear upon the present subject, we shall notice them in another place.

In the above case of ax^n, the Operation-cards merely order seven multiplications, that is, they order the mill to be in the *multiplying state* seven successive times (without any reference to the particular columns whose numbers are to be acted upon). The proper Distributive Variable-cards step in at each successive multiplication, and cause the distributions requisite for the particular case.

For	x^{an}	the operations would be			34 $(\times)$
...	$a \cdot n \cdot x$	...	...	...	$(\times, \times)$, or 2 $(\times)$
...	$\frac{a}{n} \cdot x$	...	...	...	$(\div, \times)$
...	$a + n + x$	...	...	...	$(+, +)$, or 2 $(+)$

The engine might be made to calculate all these in succession. Having completed ax^n, the function x^{an} might be written under the brackets instead of ax^n, and a new calculation commenced (the appropriate Operation and Variable-cards for the new function of course coming into play). The results would then appear on V5. So on for any number of different functions of the quantities *a*, *n*, *x*. Each *result* might either permanently remain on its column during the succeeding calculations, so that when all the functions had been computed, their values would simultaneously exist on V4, V5, V6, &c.; or each result might (after being printed off, or used in any specified manner) be effaced, to make way for its successor. The square under V4 ought, for the latter arrangement, to have the functions ax^n, x^{an}, *anx*, &c. successively inscribed in it.

Let us now suppose that we have *two* expressions whose values have been computed by the engine independently of each other (each having its own group of columns for data and results). Let them be ax^n, and *bpy*. They would then stand as follows on the columns:

V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9
+	+	+	+	+	+	+	+	+
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
a	n	x	ax^n	b	p	y	bpy	$\frac{ax^n}{bpy}$

We may now desire to combine together these two *results*, in any manner we please; in which case it would only be necessary to have an additional card or cards, which should order the requisite operations to be performed with the numbers on the two result-columns V_4 and V_8, and the *result of these further operations* to appear on a new column, V_9. Say that we wish to divide ax^n by *bpy*. The numerical value of this division would then appear on the column V_9, beneath which we have inscribed $\frac{ax^n}{bpy}$. The whole series of operations from the beginning would be as follows (*n* being = 7):

{7(×), 2(×), ÷}, or {9(×), ÷}.

This example is introduced merely to show that we may, if we please, retain separately and permanently any *intermediate* results (like ax^n, *bpy*) which occur in the course of processes having an ulterior and more complicated result as their chief and final object (like $\frac{ax^n}{bpy}$).

Any group of columns may be considered as representing a *general* function, until a *special* one has been implicitly impressed upon them through the introduction into the engine of the Operation and Variable-cards made out for a *particular* function. Thus, in the preceding example, V1, V2, V3, V5, V6, V7 represent the *general* function $\phi(a, n, b, p, x, y)$ until the function $\frac{ax^n}{bpy}$ has been determined on, and *implicitly* expressed by the placing of the right cards in the engine. The actual working of the mechanism, as regulated by these cards, then *explicitly* developes the value of the function. The inscription of a function under the brackets, and in the square under the result-column, in no way influences the processes or the results, and is merely a memorandum for the observer, to remind him of what is going on. It is the Operation and the Variable-cards only which in reality determine the function. Indeed it should be distinctly kept in mind, that the inscriptions within *any* of the squares are quite independent of the mechanism or workings of the engine, and are nothing but arbitrary memorandums placed there at pleasure to assist the spectator.

[23] See Note A.

The further we analyse the manner in which such an engine performs its processes and attains its results, the more we perceive how distinctly it places in a true and just light the mutual relations and connexion of the various steps of mathematical analysis; how clearly it separates those things which are in reality distinct and independent, and unites those which are mutually dependent.

A. A. L.

Note *C*

Those who may desire to study the principles of the Jacquard-loom in the most effectual manner, viz. that of practical observation, have only to step into the Adelaide Gallery or the Polytechnic Institution. In each of these valuable repositories of scientific *illustration*, a weaver is constantly working at a Jacquard-loom, and is ready to give any information that may be desired as to the construction and modes of acting of his apparatus. The volume on the manufacture of silk, in Lardner's Cyclopædia, contains a chapter on the Jacquard-loom, which may also be consulted with advantage.

The mode of application of the cards, as hitherto used in the art of weaving, was not found, however, to be sufficiently powerful for all the simplifications which it was desirable to attain in such varied and complicated processes as those required in order to fulfil the purposes of an Analytical Engine. A method was devised of what was technically designated *backing* the cards in certain groups according to certain laws. The object of this extension is to secure the possibility of bringing any particular card or set of cards into use *any number of times successively* in the solution of one problem. Whether this power shall be taken advantage of or not, in each particular instance, will depend on the nature of the operations which the problem under consideration may require. The process is alluded to by M. Menabrea, and it is a very important simplification. It has been proposed to use it for the reciprocal benefit of that art, which, while it has itself no apparent connexion with the domains of abstract science, has yet proved so valuable to the latter, in suggesting the principles which, in their new and singular field of application, seem likely to place *algebraical* combinations not less completely within the province of mechanism, than are all those varied intricacies of which *intersecting threads* are susceptible. By the introduction of the system of *backing* into the Jacquard-loom itself, patterns which should possess symmetry, and follow regular laws of any extent, might be woven by means of comparatively few cards.

Those who understand the mechanism of this loom will perceive that the above improvement is easily effected in practice, by causing the prism over which the train of pattern-cards is suspended to revolve *backwards* instead of *forwards*, at pleasure, under the requisite circumstances; until, by so doing, any particular card, or set of cards, that has done duty once, and passed on in the ordinary regular succession, is brought back to the position it occupied just before it was used the preceding time. The prism then resumes its *forward* rotation, and thus brings the card or set of cards in question into play a second time. This process may obviously be repeated any number of times.

A. A. L.

Note *D*

We have represented the solution of these two equations in Figure 1, with every detail, in a diagram similar to those used in Note B; but additional explanations are requisite, partly in order to make this more complicated case perfectly clear, and partly for the comprehension of certain indications and notations not used in the preceding diagrams. Those who may wish to understand Note G completely, are recommended to pay particular attention to the contents of the present Note, or they will not otherwise comprehend the similar notation and indications when applied to a much more complicated case.

Number of Operations	Nature of Operations	Variables for Data						Working Variables									Variables for Results	
		$^1V_0$	$^1V_1$	$^1V_2$	$^1V_3$	$^1V_4$	$^1V_5$	$^0V_6$	$^0V_7$	$^0V_8$	$^0V_9$	$^0V_{10}$	$^0V_{11}$	$^0V_{12}$	$^0V_{13}$	$^0V_{14}$	$^0V_{15}$	$^0V_{16}$
		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		$\boxed{m}$	$\boxed{n}$	$\boxed{d}$	$\boxed{m'}$	$\boxed{n'}$	$\boxed{d'}$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\boxed{\frac{dn'-d'n}{mn'-m'n}=x}$	$\boxed{\frac{d'm-dm}{mn'-m'n}=y}$
1	×	m	....	....	....	n'	....	mn'										
2	×	....	n	....	m'	....	....	....	$m'n$									
3	×	....	....	d	....	....	....	....	....	dn'								
4	×	....	0	....	....	....	d'	....	....	....	$d'n$							
5	×	0	....	....	....	....	0	....	....	....	....	$d'm$						
6	×	....	....	0	0	....	....	....	....	....	....	.....	dm'					
7	−	....	....	....	....	....	....	0	0	....	....	.....	.....	$(mn'-m'n)$				
8	−	....	....	....	....	....	....	....	....	0	0	.....	.....	.....	$(dn'-d'n)$			
9	−	....	....	....	....	....	....	....	....	....	....	0	0	.....	.....	$(d'm-dm')$		
10	÷	....	....	....	....	....	....	....	....	....	....	.....	.....	$(mn'-m'n)$	0	.....	$\frac{dn'-d'n}{mn'-m'n}=x$	
11	÷	....	....	....	....	....	....	....	....	....	....	.....	.....	0	.....	0	.....	$\frac{d'm-dm'}{mn'-m'n}=y$

Figure 1

In all calculations, the columns of Variables used may be divided into three classes:

- 1st. Those on which the data are inscribed.
- 2ndly. Those intended to receive the final results.
- 3rdly. Those intended to receive such intermediate and temporary combinations of the primitive data as are not to be permanently retained, but are merely needed for *working with*, in order to attain the ultimate results. Combinations of this kind might properly be called *secondary data*. They are in fact so many *successive stages* towards the final result. The columns which receive them are rightly named *Working-Variables*, for their office is in its nature purely *subsidiary* to other purposes. They develope an intermediate and transient class of results, which unite the original data with the final results.

The Result-Variables sometimes partake of the nature of Working-Variables. It frequently happens that a Variable destined to receive a final result is the recipient of one or more intermediate values successively, in the course of the processes. Similarly, the Variables for data often become Working-Variables, or Result-Variables, or even both in succession. It so happens, however, that in the case of the present equations the three sets of offices remain throughout perfectly separate and independent.

It will be observed, that in the squares below the *Working*-Variables nothing is inscribed. Any one of these Variables is in many cases destined to pass through various values successively during the performance of a calculation (although in these particular equations no instance of this occurs) . Consequently no *one fixed* symbol, or combination of symbols, should be considered as properly belonging to a merely *Working*-Variable; and as a general rule their squares are left blank. Of course in this, as in all other cases where we mention a *general* rule, it is understood that many particular exceptions may be expedient.

In order that all the indications contained in the diagram may be completely understood, we shall now explain two or three points, not hitherto touched on. When the value on any Variable is called into use, one of two consequences may be made to result. Either the value may *return* to the Variable after it has been used, in which case it is ready for a second use if needed; or the Variable may be made zero. (We are of course not considering a third case, of not unfrequent occurrence, in which the same Variable is destined to receive the *result* of the very operation which it has just supplied with a number.) Now the ordinary rule is, that the value *returns* to the Variable; unless it has been foreseen that no use for that value can recur, in which case zero is substituted. At the *end* of a calculation, therefore, every column ought as a general rule to be zero, excepting those for results. Thus it will be seen by the diagram, that when m, the value on V0, is used for the second time by Operation 5, V0 becomes 0, since m is not again needed; that similarly, when $(mn' - m'n)$, on V12, is used for the third time by Operation 11, V12 becomes zero, since $(mn' - m'n)$ is not again needed. In order to provide for the one or the other of the courses above indicated, there are *two* varieties of the *Supplying* Variable-cards. One of these varieties has provisions which cause the number given off from any Variable to *return* to that Variable after doing its duty in the mill. The other variety has provisions which cause *zero* to be substituted on the Variable, for the number given off. These two varieties are distinguished, when needful, by the respective appellations of the *Retaining* Supply-cards and the *Zero* Supply-cards. We see that the primary office (see Note B.) of both these varieties of cards is the same; they only differ in their *secondary* office.

Every Variable thus has belonging to it one class of *Receiving* Variable-cards and *two* classes of *Supplying* Variable-cards. It is plain however that only the *one* or the *other* of these two latter classes can be used by any one Variable for one operation; never *both* simultaneously, their respective functions being mutually incompatible.

It should be understood that the Variable-cards are not placed in *immediate contiguity* with the columns. Each card is connected by means of wires with the column it is intended to act upon.

Our diagram ought in reality to be placed side by side with M. Menabrea's corresponding table, so as to be compared with it, line for line belonging to each operation. But it was unfortunately inconvenient to print them in this desirable form. The diagram is, in the main, merely another manner of indicating the various relations denoted in M. Menabrea's table. Each mode has some advantages and some disadvantages. Combined, they form a complete and accurate method of registering every step and sequence in all calculations performed by the engine.

No notice has yet been taken of the *upper* indices which are added to the left of each V in the diagram; an addition which we have also taken the liberty of making to the V's in M. Menabrea's tables 3 and 4, since it does not *alter* anything therein represented by him, but merely *adds* something to the previous indications of those tables. The *lower* indices are obviously indices of *locality* only, and are wholly independent of the operations performed or of the results obtained, their value continuing unchanged during the performance of calculations. The *upper* indices, however, are of a different nature. Their office is to indicate any *alteration* in the value which a Variable represents; and they are of course liable to changes during the processes of a calculation. Whenever a Variable has only zeros upon it, it is called 0V; the moment a value appears on it (whether that value be placed there arbitrarily, or appears in the natural course of a calculation), it becomes 1V. If this value gives place to another value, the Variable becomes 2V, and so forth. Whenever a *value* again gives place to *zero*, the Variable again becomes 0V, even if it have been nV the moment before. If a *value* then again be substituted, the Variable becomes n+1V (as it would have done if it had not passed through the intermediate 0V); &c. &c. Just before any calculation is commenced, and after the data have been given, and everything adjusted and prepared for setting the mechanism in action, the upper indices of the Variables for data are all unity, and those for the Working and Result-variables are all zero. In this state the diagram represents them[24].

[24] We recommend the reader to trace the successive substitutions backwards from (1) to (4), in M. Menabrea's Table. This he will easily do by means of the upper and lower indices, and it is interesting to observe how each V successively ramifies (so to speak) into two other V's in some other column of the Table, until at length the V's of the original data are arrived at.

There are several advantages in having a set of indices of this nature; but these advantages are perhaps hardly of a kind to be immediately perceived, unless by a mind somewhat accustomed to trace the successive steps by means of which the engine accomplishes its purposes. We have only space to mention in a general way, that the whole notation of the tables is made more consistent by these indices, for they are able to mark a *difference* in certain cases, where there would otherwise be an apparent *identity* confusing in its tendency. In such a case as V*n*=V*p*+V*n* there is more clearness and more consistency with the usual laws of algebraical notation, in being able to write *m+1*V*n*=*q*V*p*+*m*V*n*. It is also obvious that the indices furnish a powerful means of tracing back the derivation of any result; and of registering various circumstances concerning that *series of successive substitutions*, of which every *result* is in fact merely the final consequence; circumstances that may in certain cases involve relations which it is important to observe, either for purely analytical reasons, or for practically adapting the workings of the engine to their occurrence. The series of substitutions which lead to the equations of the diagram are as follow:

$$\overset{(1.)}{{}^1V_{16}^{*}} = \overset{(2.)}{\frac{{}^1V_{14}}{{}^1V_{12}}} = \overset{(3.)}{\frac{{}^1V_{10}-{}^1V_{11}}{{}^1V_6-{}^1V_7}} = \overset{(4.)}{\frac{{}^1V_0.{}^1V_5-{}^1V_2.{}^1V_3}{{}^1V_0.{}^1V_4-{}^1V_3.{}^1V_1}} = \frac{d'm-dm'}{mn'-m'n}$$

$$\overset{(1.)}{{}^1V_{15}} = \overset{(2.)}{\frac{{}^1V_{13}}{{}^1V_{12}}} = \overset{(3.)}{\frac{{}^1V_8-{}^1V_9}{{}^1V_6-{}^1V_7}} = \overset{(4.)}{\frac{{}^1V_2.{}^1V_4-{}^1V_5.{}^1V_1}{{}^1V_0.{}^1V_4-{}^1V_3.{}^1V_1}} = \frac{dn'-d'n}{mn'-m'n}$$

There are *three* successive substitutions for each of these equations. The formulæ (2.), (3.) and (4.) are *implicitly* contained in (1.), which latter we may consider as being in fact the *condensed* expression of any of the former. It will be observed that every succeeding substitution must contain *twice* as many V's as its predecessor. So that if a problem require *n* substitutions, the successive series of numbers for the V's in the whole of them will be 2, 4, 8, 16…2n.

The substitutions in the preceding equations happen to be of little value towards illustrating the power and uses of the upper indices, for, owing to the nature of these particular equations, the indices are all unity throughout. We wish we had space to enter more fully into the relations which these indices would in many cases enable us to trace.

M. Menabrea incloses the three centre columns of his table under the general title *Variable-cards*. The V's however in reality all represent the actual *Variable-columns* of the engine, and not the cards that belong to them. Still the title is a very just one, since it is through the special action of certain Variable-cards (when *combined* with the more generalized agency of the Operation-cards) that every one of the particular relations he has indicated under that title is brought about.

Suppose we wish to ascertain how often any *one* quantity, or combination of quantities, is brought into use during a calculation. We easily ascertain *this*, from the inspection of any vertical column or columns of the diagram in which that quantity may appear. Thus, in the present case, we see that all the data, and all the intermediate results likewise, are used twice, excepting (*mn'* − *m'n*), which is used three times.

The *order* in which it is possible to perform the operations for the present example, enables us to effect all the eleven operations of which it consists with only *three Operation cards*; because the problem is of such a nature that it admits of each *class* of operations being performed in a group together; all the multiplications one after another, all the subtractions one after another, &c. The operations are {6(×), 3(-), 2(÷)}.

Since the very definition of an operation implies that there must be *two* numbers to act upon, there are of course *two Supplying* Variable-cards necessarily brought into action for every operation, in order to furnish the two proper numbers. (See Note B.) Also, since every operation must produce a *result*, which must be placed *somewhere*, each operation entails the action of a *Receiving* Variable-card, to indicate the proper locality for the result. Therefore, at least three times as many Variable-cards as there are *operations* (not *Operation-cards*, for these, as we have just seen, are by no means always as numerous as the *operations*) are brought into use in every calculation. Indeed, under certain contingencies, a still larger proportion is requisite; such, for example, would probably be the case when the same result has to appear on more than one Variable simultaneously (which is not unfrequently a provision necessary for subsequent purposes in a calculation), and in some other cases which we shall not here specify. We see therefore that a great disproportion exists between the amount of *Variable* and of *Operation*-cards requisite for the working of even the simplest calculation.

All calculations do not admit, like this one, of the operations of the same nature being performed in groups together. Probably very few do so without exceptions occurring in one or other stage of the progress; and some would not admit it at all. The *order* in which the operations shall be performed in every particular case is a very interesting and curious question, on which our space does not permit us fully to enter. In almost every computation a great *variety* of arrangements for the succession of the processes is possible, and various considerations must influence the selection amongst them for the purposes of a Calculating Engine. One essential object is to choose that arrangement which shall tend to reduce to a minimum the *time* necessary for completing the calculation.

It must be evident how multifarious and how mutually complicated are the considerations which the working of such an engine involve. There are frequently several distinct *sets of effects* going on simultaneously; all in a manner independent of each other, and yet to a greater or less degree exercising a mutual influence. To adjust each to every other, and indeed even to perceive and trace them out with perfect correctness and success, entails difficulties whose nature partakes to a certain extent of those involved in every question where *conditions* are very numerous and inter-complicated; such as for instance the estimation of the mutual relations amongst *statistical* phænomena, and of those involved in many other classes of facts.

A. A. L.

Note E

This example has evidently been chosen on account of its brevity and simplicity, with a view merely to explain the *manner* in which the engine would proceed in the case of an *analytical calculation containing variables*, rather than to illustrate the *extent of its powers* to solve cases of a difficult and complex nature. The equations in first example in the Memoir are in fact a more complicated problem than the present one.

We have not subjoined any diagram of its development for this new example, as we did for the former one, because this is unnecessary after the full application already made of those diagrams to the illustration of M. Menabrea's excellent tables.

It may be remarked that a slight discrepancy exists between the formulæ

$$(a + bx^1)$$
$$(\mathrm{A} + \mathrm{B}\cos^1 x)$$

given in the Memoir as the *data* for calculation, and the *results* of the calculation as developed in the last division of the table which accompanies it. To agree perfectly with this latter, the data should have been given as

$$(ax^0 + bx^1)$$
$$(\mathrm{A}\cos^0 x + \mathrm{B}\cos^1 x)$$

The following is a more complicated example of the manner in which the engine would compute a trigonometrical function containing variables. To multiply

$$\mathrm{A}+\mathrm{A}_1\cos\theta + \mathrm{A}_2\cos 2\theta + \mathrm{A}_3\cos 3\theta + \cdots$$

by $\mathrm{B} + \mathrm{B}_1\cos\theta$.

Let the resulting products be represented under the general form

$$\mathrm{C}_0 + \mathrm{C}_1\cos\theta + \mathrm{C}_2\cos 2\theta + \mathrm{C}_3\cos 3\theta + \cdots \qquad (1.)$$

This trigonometrical series is not only in itself very appropriate for illustrating the processes of the engine, but is likewise of much practical interest from its frequent use in astronomical computations. Before proceeding further with it, we shall point out that there are three very distinct classes of ways in which it may be desired to deduce numerical values from any analytical formula.

First. We may wish to find the collective numerical value of the *whole formula*, without any reference to the quantities of which that formula is a function, or to the particular mode of their combination and distribution, of which the formula is the result and representative. Values of this kind are of a strictly arithmetical nature in the most limited sense of the term, and retain no trace whatever of the processes through which they have been deduced. In fact, any one such numerical value may have been attained from an *infinite variety* of data, or of problems. The values for *x* and *y* in the two equations (see Note D.) come under this class of numerical results.

Secondly. We may propose to compute the collective numerical value of *each term* of a formula, or of a series, and to keep these results separate. The engine must in such a case appropriate as many columns to *results* as there are terms to compute.

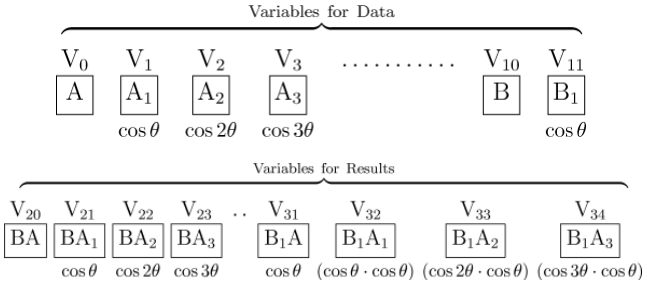
Thirdly. It may be desired to compute the numerical value of various *subdivisions of each term*, and to keep all these results separate. It may be required, for instance, to compute each coefficient separately from its variable, in which particular case the engine must appropriate *two* result-columns to *every term that contains both a variable and coefficient*.

There are many ways in which it may be desired in special cases to distribute and keep separate the numerical values of different parts of an algebraical formula; and the power of effecting such distributions to any extent is essential to the *algebraical* character of the Analytical Engine. Many persons who are not conversant with mathematical studies, imagine that because the business of the engine is to give its results in *numerical notation*, the *nature of its processes* must consequently be *arithmetical* and *numerical*, rather than *algebraical* and *analytical*. This is an error. The engine can arrange and combine its numerical quantities exactly as if they were *letters* or any other *general* symbols; and in fact it might bring out its results in algebraical *notation*, were provisions made accordingly. It might develope three sets of results simultaneously, viz. *symbolic* results (as already alluded to in Notes A. and B.), *numerical* results (its chief and primary object); and *algebraical* results in *literal* notation. This latter however has not been deemed a necessary or desirable addition to its powers, partly because the necessary arrangements for effecting it would increase the complexity and extent of the mechanism to a degree that would not be commensurate with the advantages, where the main object of the invention is to translate into *numerical* language general formulæ of analysis already known to us, or whose laws of formation are known to us. But it would be a mistake to suppose that because its *results* are given in the *notation* of a more restricted science, its *processes* are therefore restricted to those of that science. The object of the engine is in fact to give the *utmost practical efficiency* to the resources of *numerical interpretations* of the higher science of analysis, while it uses the processes and combinations of this latter.

To return to the trigonometrical series. We shall only consider the first four terms of the factor (A + A1 cos θ+ &c.), since this will be sufficient to show the method. We propose to obtain separately the numerical value of each coefficient C0, C1, &c. of (1.). The direct multiplication of the two factors gives

$$\left.\begin{array}{llllll} \mathrm{BA} & + \mathrm{BA}_1\cos\theta & + \mathrm{BA}_2\cos 2\theta & + \mathrm{BA}_3\cos 3\theta & + \ldots\ldots\ldots\ldots \\ & \mathrm{B}_1\mathrm{A}\cos\theta & + \mathrm{B}_1\mathrm{A}_1\cos\theta\cdot\cos\theta & + \mathrm{B}_1\mathrm{A}_2\cos 2\theta\cdot\cos\theta & + \mathrm{B}_1\mathrm{A}_3\cos 3\theta\cdot\cos\theta \end{array}\right\} \quad (2)$$

a result which would stand thus on the engine:



The variable belonging to each coefficient is written below it, as we have done in the diagram, by way of memorandum. The only further reduction which is at first apparently possible in the preceding result, would be the addition of V21 to V31 (in which case B1A should be effaced from V31). The whole operations from the beginning would then be

First Series of Operations	*Second Series of Operations*	*Third Series, which contains only one (final) operation*
$^1V_{10}\times{}^1V_0 = {}^1V_{20}$	$^1V_{11}\times{}^1V_0 = {}^1V_{31}$	$^1V_{21}\times{}^1V_{31} = {}^2V_{21}$, and
$^1V_{10}\times{}^1V_1 = {}^1V_{21}$	$^1V_{11}\times{}^1V_1 = {}^1V_{32}$	V_{31} becomes = 0.
$^1V_{10}\times{}^1V_2 = {}^1V_{22}$	$^1V_{11}\times{}^1V_2 = {}^1V_{33}$	
$^1V_{10}\times{}^1V_3 = {}^1V_{23}$	$^1V_{11}\times{}^1V_3 = {}^1V_{34}$	

We do not enter into the same detail of *every* step of the processes as in the examples of Notes D. and G., thinking it unnecessary and tedious to do so. The reader will remember the meaning and use of the upper and lower indices, &c., as before explained.

To proceed: we know that

$$\cos n\theta \cdot \cos\theta = \tfrac{1}{2}\cos\overline{n+1}\theta + \tfrac{1}{2}\overline{n-1}\theta \quad (3.)$$

Consequently, a slight examination of the second line of (2.) will show that by making the proper substitutions, (2.) will become

BA	$+\mathrm{BA}_1 \cdot \cos\theta$	$+\mathrm{BA}_2 \cdot \cos 2\theta$	$+\mathrm{BA}_3 \cdot \cos 3\theta$	
$+\tfrac{1}{2}\mathrm{B}_1\mathrm{A}_1$	$+\mathrm{B}_1\mathrm{A} \cdot \cos\theta$	$+\tfrac{1}{2}\mathrm{B}_1\mathrm{A}_1 \cdot \cos 2\theta$		
	$+\tfrac{1}{2}\mathrm{B}_1\mathrm{A}_2 \cdot \cos\theta$	$+\tfrac{1}{2}\mathrm{B}_1\mathrm{A}_3 \cdot \cos 2\theta$	$+\tfrac{1}{2}\mathrm{B}_1\mathrm{A}_2 \cdot \cos 3\theta$	$+\tfrac{1}{2}\mathrm{B}_1\mathrm{A}_3 \cdot \cos 4\theta$
C_0	C_1	C_2	C_3	C_4

These coefficients should respectively appear on

$$\mathrm{V}_{20} \qquad \mathrm{V}_{21} \qquad \mathrm{V}_{22} \qquad \mathrm{V}_{23} \qquad \mathrm{V}_{24}$$

We shall perceive, if we inspect the particular arrangement of the results in (2.) on the Result-columns as represented in the diagram, that, in order to effect this transformation, each successive coefficient upon V_{32}, V_{33}, &c. (beginning with V_{32}), must through means of proper cards be divided by two[25]; and that one of the halves thus obtained must be added to the coefficient on the Variable which precedes it by ten columns, and the other half to the coefficient on the Variable which precedes it by twelve columns; V_{32}, V_{33}, &c. themselves becoming zeros during the process.

This series of operations may be thus expressed[26]:

Fourth Series

$$\left\{\begin{array}{lllll} {}^1\mathrm{V}_{32} \div 2 + {}^1\mathrm{V}_{22} & = & {}^2\mathrm{V}_{22} & = & \mathrm{BA}_2 + \tfrac{1}{2}\mathrm{B}_1\mathrm{A}_1 \\ {}^1\mathrm{V}_{32} \div 2 + {}^1\mathrm{V}_{20} & = & {}^2\mathrm{V}_{20} & = & \mathrm{BA} + \tfrac{1}{2}\mathrm{B}_1\mathrm{A}_1 \ldots\ldots\ldots\ldots = \mathrm{C}_0 \end{array}\right.$$

$$\left\{\begin{array}{lllll} {}^1\mathrm{V}_{33} \div 2 + {}^1\mathrm{V}_{23} & = & {}^2\mathrm{V}_{23} & = & \mathrm{BA}_3 + \tfrac{1}{2}\mathrm{B}_1\mathrm{A}_2 \ldots\ldots\ldots\ldots = \mathrm{C}_3\dagger \\ {}^1\mathrm{V}_{33} \div 2 + {}^2\mathrm{V}_{21} & = & {}^3\mathrm{V}_{21} & = & \mathrm{BA}_1 + \mathrm{B}_1\mathrm{A} + \tfrac{1}{2}\mathrm{B}_1\mathrm{A}_2 \ldots = \mathrm{C}_1 \end{array}\right.$$

$$\left\{\begin{array}{lllll} {}^1\mathrm{V}_{34} \div 2 + {}^0\mathrm{V}_{24} & = & {}^1\mathrm{V}_{24} & = & \tfrac{1}{2}\mathrm{B}_1\mathrm{A}_3 \ldots\ldots\ldots\ldots\ldots\ldots = \mathrm{C}_4 \\ {}^1\mathrm{V}_{34} \div 2 + {}^2\mathrm{V}_{22} & = & {}^3\mathrm{V}_{22} & = & \mathrm{BA}_2 + \tfrac{1}{2}\mathrm{B}_1\mathrm{A}_1 + \tfrac{1}{2}\mathrm{B}_1\mathrm{A}_3 \quad = \mathrm{C}_2. \end{array}\right.$$

The calculation of the coefficients C_0, C_1, &c. of (1.) would now be completed, and they would stand ranged in order on V_{20}, V_{21}, &c. It will be remarked, that from the moment the fourth series of operations is ordered, the Variables V_{31}, V_{32}, &c. cease to be *Result*-Variables, and become mere *Working*-Variables.

The substitution made by the engine of the processes in the second side of (3.) for those in the first side is an excellent illustration of the manner in which we may arbitrarily order it to substitute any function, number, or process, at pleasure, for any other function, number or process, on the occurrence of a specified contingency.

We will now suppose that we desire to go a step further, and to obtain the numerical value of each *complete* term of the product (1.); that is, of each *coefficient and variable united*, which for the (n + 1)th term would be $\mathrm{C}_n \cdot \cos n\theta$.

We must for this purpose place the variables themselves on another set of columns, V_{41}, V_{42}, &c., and then order their successive multiplication by V_{21}, V_{22}, &c., each for each. There would thus be a final series of operations as follows:

Fifth and Final Series of Operations

$$\begin{array}{l} {}^2\mathrm{V}_{20} \times {}^0\mathrm{V}_{40} = {}^1\mathrm{V}_{40} \\ {}^3\mathrm{V}_{21} \times {}^0\mathrm{V}_{41} = {}^1\mathrm{V}_{41} \\ {}^3\mathrm{V}_{22} \times {}^0\mathrm{V}_{42} = {}^1\mathrm{V}_{42} \\ {}^2\mathrm{V}_{23} \times {}^0\mathrm{V}_{43} = {}^1\mathrm{V}_{43} \\ {}^1\mathrm{V}_{24} \times {}^0\mathrm{V}_{44} = {}^1\mathrm{V}_{44} \end{array}$$

(N.B. that V_{40} being intended to receive the coefficient on V_{20} which has *no* variable, will only have cos 0θ (=1) inscribed on it, preparatory to commencing the fifth series of operations.)

From the moment that the fifth and final series of operations is ordered, the Variables V_{20}, V_{21}, &c. then in their turn cease to be *Result*-Variables and become mere *Working*-Variables; V_{40}, V_{41}, &c. being now the recipients of the ultimate results.

We should observe, that if the variables cos θ, cos 2θ, cos 3θ, &c. are furnished, they would be placed directly upon V_{41}, V_{42}, &c., like any other data. If not, a separate computation might be entered upon in a separate part of the engine, in order to calculate them, and place them on V_{41}, &c.

We have now explained how the engine might compute (1.) in the most direct manner, supposing we knew nothing about the *general* term of the resulting series. But the engine would in reality set to work very differently, whenever (as in this case) we *do* know the law for the general term.

The first two terms of (1.) are

$$(\mathrm{BA} + \tfrac{1}{2}\mathrm{B}_1\mathrm{A}_1) + (\overline{\mathrm{BA}_1 + \mathrm{B}_1\mathrm{A} + \tfrac{1}{2}\mathrm{B}_1\mathrm{A}_2} \cdot \cos\theta) \quad (4.)$$

and the general term for all after these is

$$(\mathrm{BA}_n + \tfrac{1}{2}\mathrm{B}_1 \cdot \overline{\mathrm{A}_{n-1} + \mathrm{A}_{n+2}})\cos n\theta \quad (5.)$$

which is the coefficient of the (n+1)th term. The engine would calculate the first two terms by means of a separate set of suitable Operation-cards, and would then need another set for the third term; which last set of Operation-cards would calculate all the succeeding terms *ad infinitum*, merely requiring certain new Variable-cards for each term to direct the operations to act on the proper columns. The following would be the successive sets of operations for computing the coefficients of n+2 terms:

(×, ×, ÷, +), (×, ×, ×, ÷, +, +), n(×, +, ×, ÷, +).

Or we might represent them as follows, according to the numerical order of the operations:

(1, 2…4), (5, 6…10), n(11, 12…15).

The brackets, it should be understood, point out the relation in which the operations may be *grouped*, while the comma marks *succession*. The symbol + might be used for this latter purpose, but this would be liable to produce confusion, as + is also

[25] This division would be managed by ordering the number 2 to appear on any separate new column which should be conveniently situated for the purpose, and then directing this column (which is in the strictest sense a *Working*-Variable) to divide itself successively with V_{32}, V_{33}, &c.

[26] It should be observed, that were the rest of the factor (A + A cos θ + &c.) taken into account, instead of *four* terms only, C_3 would have the additional term $\frac{1}{2}\mathrm{B}_1\mathrm{A}_4$; and C_4 the two additional terms, BA_4, $\frac{1}{2}\mathrm{B}_1\mathrm{A}_5$. This would indeed have been the case had even *six* terms been multiplied.

necessarily used to represent one class of the actual operations which are the subject of that succession. In accordance with this meaning attached to the comma, care must be taken when any one group of operations recurs more than once, as is represented above by *n*(11…l5), not to insert a comma after the number or letter prefixed to that group. *n*, (11…15) would stand for *an operation n, followed by the group of operations* (11…15); instead of denoting *the number of groups which are to follow each other*.

Wherever a *general* term exists, there will be a *recurring group* of operations, as in the above example. Both for brevity and for distinctness, a *recurring group* is called a *cycle*. A *cycle* of operations, then, must be understood to signify any *set of operations* which is repeated *more than once*. It is equally a *cycle*, whether it be repeated *twice* only, or an indefinite number of times; for it is the fact of a *repetition occurring at all* that constitutes it such. In many cases of analysis there is a *recurring group* of one or more *cycles*; that is, a *cycle of a cycle*, or a *cycle of cycles*. For instance: suppose we wish to divide a series by a series,

(1.) $$\frac{a+bx+cx^2+\ldots}{a'+b'x+c'x^2+\ldots},$$

it being required that the result shall be developed, like the dividend and the divisor, in successive powers of *x*. A little consideration of (1.), and of the steps through which algebraical division is effected, will show that (if the denominator be supposed to consist of *p* terms) the first partial quotient will be completed by the following operations:—

(2.) $$\{(\div), p(\times, -)\} \text{ or } \{(1), p(2, 3)\},$$

that the second partial quotient will be completed by an exactly similar set of operations, which acts on the remainder obtained by the first set, instead of on the original dividend. The whole of the processes therefore that have been gone through, by the time the *second* partial quotient has been obtained, will be,—

(3.) $$2\{(\div), p(\times, -)\} \text{ or } 2\{(1), p(2, 3)\},$$

which is a cycle that includes a cycle, or a cycle of the second order. The operations for the *complete* division, supposing we propose to obtain *n* terms of the series constituting the quotient, will be,—

(4.) $$n\{(\div), p(\times, -)\} \text{ or } n\{(1), p(2, 3)\},$$

It is of course to be remembered that the process of algebraical division in reality continues *ad infinitum*, except in the few exceptional cases which admit of an exact quotient being obtained. The number *n* in the formula (4.) is always that of the number of terms we propose to ourselves to obtain; and the *n*th partial quotient is the coefficient of the (*n*-1)th power of *x*.

There are some cases which entail *cycles of cycles of cycles*, to an indefinite extent. Such cases are usually very complicated, and they are of extreme interest when considered with reference to the engine. The algebraical development in a series of the *n*th function of any given function is of this nature. Let it be proposed to obtain the *n*th function of

(5.) $\phi(a, b, c, \ldots, x)$, *x* being the variable.

We should premise, that we suppose the reader to understand what is meant by an *n*th function. We suppose him likewise to comprehend distinctly the difference between developing *an n*th *function algebraically*, and merely *calculating an n*th *function arithmetically*. If he does not, the following will be by no means very intelligible; but we have not space to give any preliminary explanations. To proceed: the law, according to which the successive functions of (5.) are to be developed, must of course first be fixed on. This law may be of very various kinds. We may propose to obtain our results in successive *powers* of *x*, in which case the general form would be

$$C + C_1x + C_2x^2 + \&c.;$$

or in successive powers of *n* itself, the index of the function we are ultimately to obtain, in which case the general form would be

$$C + C_1n + C_2n^2 + \&c.;$$

and *x* would only enter in the coefficients. Again, other functions of *x* or of *n* instead of *powers* might be selected. It might be in addition proposed, that the coefficients themselves should be arranged according to given functions of a certain quantity. Another mode would be to make equations arbitrarily amongst the coefficients only, in which case the several functions, according to either of which it might be possible to develope the *n*th function of (5.), would have to be determined from the combined consideration of these equations and of (5.) itself.

The *algebraical* nature of the engine (so strongly insisted on in a previous part of this Note) would enable it to follow out any of these various modes indifferently; just as we recently showed that it can distribute and separate the numerical results of any one prescribed series of processes, in a perfectly arbitrary manner. Were it otherwise, the engine could merely *compute the arithmetical n*th *function*, a result which, like any other purely arithmetical results, would be simply a collective number, bearing no traces of the data or the processes which had led to it.

Secondly, the *law* of development for the *n*th function being selected, the next step would obviously be to develope (5.) itself, according to this law. This result would be the first function, and would be obtained by a determinate series of processes. These in most cases would include amongst them one or more *cycles* of operations.

The third step (which would consist of the various processes necessary for effecting the actual substitution of the series constituting the *first function*, for the *variable* itself) might proceed in either of two ways. It might make the substitution either wherever *x* occurs in the original (5.), or it might similarly make it wherever *x* occurs in the first function itself which is the equivalent of (5.). In some cases the former mode might be best, and in others the latter.

Whichever is adopted, it must be understood that the result is to appear arranged in a series following the law originally prescribed for the development of the *n*th function. This result constitutes the second function; with which we are to proceed exactly as we did with the first function, in order to obtain the third function, and so on, *n*-1 times, to obtain the *n*th function. We easily perceive that since every successive function is arranged in a series *following the same law*, there would (after the *first* function is obtained) be a

cycle of a cycle of a cycle, &c. of operations[27], one, two, three, up to *n*-1 times, in order to get the *n*th function. We say, *after the first function is obtained*, because (for reasons on which we cannot here enter) the *first* function might in many cases be developed through a set of processes peculiar to itself, and not recurring for the remaining functions.

We have given but a very slight sketch of the principal *general* steps which would be requisite for obtaining an *n*th function of such a formula as (5.). The question is so exceedingly complicated, that perhaps few persons can be expected to follow, to their own satisfaction, so brief and general a statement as we are here restricted to on this subject. Still it is a very important case as regards the engine, and suggests ideas peculiar to itself, which we should regret to pass wholly without allusion. Nothing could be more interesting than to follow out, in every detail, the solution by the engine of such a case as the above; but the time, space and labour this would necessitate, could only suit a very extensive work.

To return to the subject of *cycles* of operations: some of the notation of the integral calculus lends itself very aptly to express them: (2.) might be thus written:—

(6.) $$(\div), \sum(+1)^p(\times, -) \quad \text{or} \quad (1), \sum(+1)^p(2, 3),$$

where *p* stands for the variable; (+ 1)*p* for the function of the variable, that is, for ϕp; and the limits are from 1 to *p*, or from 0 to *p*-1, each increment being equal to unity. Similarly, (4.) would be,

(7.) $$\sum(+1)^n\{(\div), \sum(+1)^p(\times, -)\}$$

the limits of n being from 1 to *n*, or from 0 to *n*-1,

(8.) $$\text{or } \sum(+1)^n\{(1), \sum(+1)^p(2, 3)\}.$$

Perhaps it may be thought that this notation is merely a circuitous way of expressing what was more simply and as effectually expressed before; and, in the above example, there may be some truth in this. But there is another description of cycles which *can* only effectually be expressed, in a condensed form, by the preceding notation. We shall call them *varying cycles*. They are of frequent occurrence, and include successive cycles of operations of the following nature:—

(9.) $$p(1, 2 \dots m), \overline{p-1}(1, 2 \dots m), \overline{p-2}(1, 2 \dots m) \dots \overline{p-n}(1, 2 \dots m),$$

where each cycle contains the same group of operations, but in which the number of repetitions of the group varies according to a fixed rate, with every cycle. (9.) can be well expressed as follows:—

(10.) $\sum p(1, 2 \dots m)$, the limits of *p* being from *p*-*n* to *p*.

[27] A cycle that includes *n* other cycles, successively *contained one within another*, is called a cycle of the *n*+1th order. A cycle may simply *include* many other cycles, and yet only be of the second order. If a series follows a certain law for a certain number of terms, and then another law for another number of terms, there will be a cycle of operations for every new law; but these cycles will not be *contained one within another*,—they merely *follow each other*. Therefore their number may be infinite without influencing the *order* of a cycle that includes a repetition of such a series.

Independent of the intrinsic advantages which we thus perceive to result in certain cases from this use of the notation of the integral calculus, there are likewise considerations which make it interesting, from the connections and relations involved in this new application. It has been observed in some of the former Notes, that the processes used in analysis form a logical system of much higher generality than the applications to number merely. Thus, when we read over any algebraical formula, considering it exclusively with reference to the processes of the engine, and putting aside for the moment its abstract signification as to the relations of quantity, the symbols +, ×, &c. in reality represent (as their immediate and proximate effect, when the formula is applied to the engine) that a certain prism which is a part of the mechanism (see Note C.) turns a new face, and thus presents a new card to act on the bundles of levers of the engine; the new card being perforated with holes, which are arranged according to the peculiarities of the operation of addition, or of multiplication, &c. Again, the *numbers* in the preceding formula (8.), each of them really represents one of these very pieces of card that are hung over the prism.

Now in the use made in the formulæ (7.), (8.) and (10.), of the notation of the integral calculus, we have glimpses of a similar new application of the language of the *higher* mathematics. $\sum$, in reality, here indicates that when a certain number of cards have acted in succession, the prism over which they revolve must *rotate backwards*, so as to bring those cards into their former position; and the limits 1 to *n*, 1 to *p*, &c., regulate how often this backward rotation is to be repeated.

A. A. L.

Note F

There is in existence a beautiful woven portrait of Jacquard, in the fabrication of which 24,000 cards were required.

The power of *repeating* the cards, alluded to by M. Menabrea, and more fully explained in Note C., reduces to an immense extent the number of cards required. It is obvious that this mechanical improvement is especially applicable wherever *cycles* occur in the mathematical operations, and that, in preparing data for calculations by the engine, it is desirable to arrange the order and combination of the processes with a view to obtain them as much as possible *symmetrically* and in cycles, in order that the mechanical advantages of the *backing* system may be applied to the utmost. It is here interesting to observe the manner in which the value of an *analytical* resource is *met* and *enhanced* by an ingenious *mechanical* contrivance. We see in it an instance of one of those mutual *adjustments* between the purely mathematical and the mechanical departments, mentioned in Note A. as being a main and essential condition of success in the invention of a calculating engine. The nature of the resources afforded by such adjustments would be of two principal kinds. In some cases, a difficulty (perhaps in itself insurmountable) in the one department would be overcome by facilities in the other; and sometimes (as in the present case) a strong point in the one would be rendered still stronger and more available by combination with a corresponding strong point in the other.

As a mere example of the degree to which the combined systems of cycles and of backing can diminish the *number* of cards requisite, we shall choose a case which places it in strong

evidence, and which has likewise the advantage of being a perfectly different *kind* of problem from those that are mentioned in any of the other Notes. Suppose it be required to eliminate nine variables from ten simple equations of the form—

$$ax_0 + bx_1 + cx_2 + dx_3 + \cdots = p \quad (1.)$$
$$a^1x_0 + b^1x_1 + c^1x_2 + d^1x_3 + \cdots = p' \quad (2.)$$
&c. &c. &c. &c.

We should explain, before proceeding, that it is not our object to consider this problem with reference to the actual arrangement of the data on the Variables of the engine, but simply as an abstract question of the *nature* and *number* of the *operations* required to be performed during its complete solution.

The first step would be the elimination of the first unknown quantity $x0$ between the first two equations. This would be obtained by the form

$$(a^1a\text{-}aa^1)x_0 + (a^1b\text{-}ab^1)x_1 + (a^1c\text{-}ac^1)x_2 + \\ + (a^1d\text{-}ad^1)x_3 + \cdots = a^1p\text{-}ap^1,$$

for which the operations 10 (×, ×, −) would be needed. The second step would be the elimination of $x0$ between the second and third equations, for which the operations would be precisely the same. We should then have had altogether the following operations:—

$$10(\times, \times, -),\ 10(\times, \times, -) = 20(\times, \times, -)$$

Continuing in the same manner, the total number of operations for the complete elimination of $x0$ between all the successive pairs of equations would be

$$9 \cdot 10(\times, \times, -) = 90(\times, \times, -)$$

We should then be left with nine simple equations of nine variables from which to eliminate the next variable $x1$, for which the total of the processes would be

$$8 \cdot 9(\times, \times, -) = 72(\times, \times, -)$$

We should then be left with eight simple equations of eight variables from which to eliminate $x2$, for which the processes would be—

$$7 \cdot 8(\times, \times, -) = 56(\times, \times, -)$$

and so on. The total operations for the elimination of all the variables would thus be—

$$9{\cdot}10 + 8{\cdot}9 + 7{\cdot}8 + 6{\cdot}7 + 5{\cdot}6 + 4{\cdot}5 + 3{\cdot}4 + 2{\cdot}3 + 1{\cdot}2 = 330.$$

So that *three* Operation-cards would perform the office of 330 such cards.

If we take n simple equations containing $n-1$ variables, n being a number unlimited in magnitude, the case becomes still more obvious, as the same three cards might then take the place of thousands or millions of cards.

We shall now draw further attention to the fact, already noticed, of its being by no means necessary that a formula proposed for solution should ever have been actually worked out, as a condition for enabling the engine to solve it. Provided we know the *series of operations* to be gone through, that is sufficient. In the foregoing instance this will be obvious enough on a slight consideration. And it is a circumstance which deserves particular notice, since herein may reside a latent value of such an engine almost incalculable in its possible ultimate results. We already know that there are functions whose numerical value it is of importance for the purposes both of abstract and of practical science to ascertain, but whose determination requires processes so lengthy and so complicated, that, although it is possible to arrive at them through great expenditure of time, labour and money, it is yet on these accounts practically almost unattainable; and we can conceive there being some results which it may be *absolutely impossible* in practice to attain with any accuracy, and whose precise determination it may prove highly important for some of the future wants of science, in its manifold, complicated and rapidly-developing fields of inquiry, to arrive at.

Without, however, stepping into the region of conjecture, we will mention a particular problem which occurs to us at this moment as being an apt illustration of the use to which such an engine may be turned for determining that which human brains find it difficult or impossible to work out unerringly. In the solution of the famous problem of the Three Bodies, there are, out of about 295 coefficients of lunar perturbations given by M. Clausen (Astroe. Nachrichten, No. 406) as the result of the calculations by Burg, of two by Damoiseau, and of one by Burckhardt, fourteen coefficients that differ in the nature of their algebraic sign; and out of the remainder there are only 101 (or about one-third) that agree precisely both in signs and in amount. These discordances, which are generally small in individual magnitude, may arise either from an erroneous determination of the abstract coefficients in the development of the problem, or from discrepancies in the data deduced from observation, or from both causes combined. The former is the most ordinary source of error in astronomical computations, and this the engine would entirely obviate.

We might even invent laws for series or formulæ in an arbitrary manner, and set the engine to work upon them, and thus deduce numerical results which we might not otherwise have thought of obtaining; but this would hardly perhaps in any instance be productive of any great practical utility, or calculated to rank higher than as a philosophical amusement.

A. A. L.

Note G

It is desirable to guard against the possibility of exaggerated ideas that might arise as to the powers of the Analytical Engine. In considering any new subject, there is frequently a tendency, first, to *overrate* what we find to be already interesting or remarkable; and, secondly, by a sort of natural reaction, to *undervalue* the true state of the case, when we do discover that our notions have surpassed those that were really tenable.

The Analytical Engine has no pretensions whatever to *originate* anything. It can do whatever we *know how to order it* to perform. It can *follow* analysis; but it has no power of *anticipating* any analytical relations or truths. Its province is to assist us in making *available* what we are already acquainted with. This it is calculated to effect primarily and chiefly of course, through its executive faculties; but it is likely to exert an *indirect* and reciprocal influence on science itself in another manner. For, in so distributing and combining the truths and the formulæ of analysis, that they may become most easily and rapidly amenable to the mechanical combinations of the engine, the relations and the nature of many subjects in that science are necessarily thrown into

new lights, and more profoundly investigated. This is a decidedly indirect, and a somewhat *speculative*, consequence of such an invention. It is however pretty evident, on general principles, that in devising for mathematical truths a new form in which to record and throw themselves out for actual use, views are likely to be induced, which should again react on the more theoretical phase of the subject. There are in all extensions of human power, or additions to human knowledge, various *collateral* influences, besides the main and primary object attained.

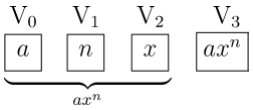
To return to the executive faculties of this engine: the question must arise in every mind, are they *really* even able to *follow* analysis in its whole extent? No reply, entirely satisfactory to all minds, can be given to this query, excepting the actual existence of the engine, and actual experience of its practical results. We will however sum up for each reader's consideration the chief elements with which the engine works:

1. It performs the four operations of simple arithmetic upon any numbers whatever.
2. By means of certain artifices and arrangements (upon which we cannot enter within the restricted space which such a publication as the present may admit of), there is no limit either to the *magnitude* of the *numbers* used, or to the *number of quantities* (either variables or constants) that may be employed.
3. It can combine these numbers and these quantities either algebraically or arithmetically, in relations unlimited as to variety, extent, or complexity.
4. It uses algebraic *signs* according to their proper laws, and developes the logical consequences of these laws.
5. It can arbitrarily substitute any formula for any other; effacing the first from the columns on which it is represented, and making the second appear in its stead.
6. It can provide for singular values. Its power of doing this is referred to in M. Menabrea's memoir, where he mentions the passage of values through zero and infinity. The practicability of causing it arbitrarily to change its processes at any moment, on the occurrence of any specified contingency (of which its substitution of $(\frac{1}{2}\cos\overline{n+1}\theta + \frac{1}{2}\cos\overline{n-1}\theta)$ for $(\cos n\theta \cdot \cos\theta)$, explained in Note E, is in some degree an illustration), at once secures this point.

The subject of integration and of differentiation demands some notice. The engine can effect these processes in either of two ways:

First. We may order it, by means of the Operation and of the Variable-cards, to go through the various steps by which the required *limit* can be worked out for whatever function is under consideration.

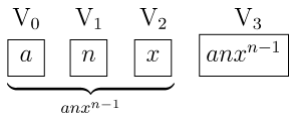
Secondly. It may (if we know the form of the limit for the function in question) effect the integration or differentiation by direct[28] substitution. We remarked in Note B., that any *set* of columns on which numbers are inscribed, represents merely a *general* function of the several quantities, until the special function have been impressed by means of the Operation and Variable-cards. Consequently, if instead of requiring the value of the function, we require that of its integral, or of its differential coefficient, we have merely to order whatever particular combination of the ingredient quantities may constitute that integral or that coefficient. In ax^n, for instance, instead of the quantities



being ordered to appear on V3 in the combination ax^n, they would be ordered to appear in that of

$$anx^{n-1}$$

They would then stand thus:



Similarly, we might have $\frac{a}{n+1}x^{n+1}$, the integral of ax^n.

An interesting example for following out the processes of the engine would be such a form as

$$\int \frac{x^n\,dx}{\sqrt{a^2-x^2}},$$

or any other cases of integration by successive reductions, where an integral which contains an operation repeated *n* times can be made to depend upon another which contains the same *n*-1 or *n*-2 times, and so on until by continued reduction we arrive at a certain *ultimate* form, whose value has then to be determined.

The methods in Arbogast's *Calcul des Dérivations* are peculiarly fitted for the notation and the processes of the engine. Likewise the whole of the Combinatorial Analysis, which consists first in a purely numerical calculation of indices, and secondly in the distribution and combination of the quantities according to laws prescribed by these indices.

We will terminate these Notes by following up in detail the steps through which the engine could compute the Numbers of Bernoulli, this being (in the form in which we shall deduce it) a rather complicated example of its powers. The simplest manner of computing these numbers would be from the direct expansion of

[28] The engine cannot of course compute limits for perfectly *simple* and *uncompounded* functions, except in this manner. It is obvious that it has no power of representing or of manipulating with any but *finite* increments or decrements, and consequently that wherever the computation of limits (or of any other functions) depends upon the *direct* introduction of quantities which either increase or decrease *indefinitely*, we are absolutely beyond the sphere of its powers. Its nature and arrangements are remarkably adapted for taking into account all *finite* increments or decrements (however small or large), and for developing the true and logical modifications of form or value dependent upon differences of this nature. The engine may indeed be considered as including the whole Calculus of Finite Differences; many of whose theorems would be especially and beautifully fitted for development by its processes, and would offer peculiarly interesting considerations. We may mention, as an example the calculation of the Numbers of Bernoulli by means of the *Differences of Zero*.

$$\frac{x}{\epsilon^x - 1} = \frac{1}{1 + \frac{x}{2} + \frac{x^2}{2\cdot3} + \frac{x^3}{2\cdot3\cdot4} + \&\text{c.}} \quad (1.)$$

which is in fact a particular case of the development of

$$\frac{a + bx + cx^2 + \&\text{c.}}{a' + b'x + c'x^2 + \&\text{c.}}$$

mentioned in Note E. Or again, we might compute them from the well-known form

$$\mathrm{B}_{2n-1} = 2 \cdot \frac{1\cdot2\cdot3\ldots2n}{(2\pi)^{2n}} \cdot \left\{1 + \frac{1}{2^{2n}} + \frac{1}{3^{2n}} + \cdots\right\} \quad (2.)$$

or from the form

$$\mathrm{B}_{2n-1} = \frac{\pm 2^n}{(2^{2n}-1)2^{n-1}} \left\{ \begin{array}{lll} & \frac{1}{2}n^{2n-1} & \\ - & (n-1)^{2n-1}\left\{1 + \frac{1}{2}\cdot\frac{2n}{1}\right\} & \\ + & (n-2)^{2n-1}\left\{1 + \frac{2n}{1} + \frac{1}{2}\cdot\frac{2n\cdot(2n-1)}{1\cdot2}\right\} & \\ - & (n-3)^{2n-1}\left\{ \begin{array}{l} 1 \quad + \frac{2n}{1} + \frac{2n\cdot(2n-1)}{1\cdot2} + \\ \qquad + \frac{1}{2}\cdot\frac{2n\cdot(2n-1)\cdot(2n-2)}{1\cdot2\cdot3} \end{array} \right\} & \\ + & \cdots \qquad \cdots \qquad \cdots & \cdots \end{array} \right\} \quad (3.)$$

or from many others. As however our object is not simplicity or facility of computation, but the illustration of the powers of the engine, we prefer selecting the formula below, marked (8.) This is derived in the following manner.

If in the equation

$$\frac{x}{\epsilon^x - 1} = 1 - \frac{x}{2} + \mathrm{B}_1\frac{x^2}{2} + \mathrm{B}_3\frac{x^4}{2\cdot3\cdot4} + \mathrm{B}_5\frac{x^6}{2\cdot3\cdot4\cdot5\cdot6} + \cdots \quad (4.)$$

(in which B_1, B_3…, &c. are the Numbers of Bernoulli), we expand the denominator of the first side in powers of *x*, and then divide both numerator and denominator by *x*, we shall derive

$$1 = \left(1 - \frac{x}{2} + \mathrm{B}_1\frac{x^2}{2} + \mathrm{B}_3\frac{x^4}{2\cdot3\cdot4} + \cdots\right)\left(1 + \frac{x}{2} + \frac{x^2}{2\cdot3} + \frac{x^3}{2\cdot3\cdot4} + \cdots\right) \quad (5.)$$

If this latter multiplication be actually performed, we shall have a series of the general form

$$1 + \mathrm{D}_1x + \mathrm{D}_2x^2 + \mathrm{D}_3x^3 + \cdots \quad (6.)$$

in which we see, first, that all the coefficients of the powers of *x* are severally equal to zero; and secondly, that the general form for D_{2n}, the coefficient of the 2*n*+1th *term* (that is of x^{2n} any *even* power of *x*), is the following:

$$\left.\begin{array}{l} \frac{1}{2\cdot3\ldots2n+1} - \frac{1}{2}\cdot\frac{1}{2\cdot3\ldots2n} + \frac{\mathrm{B}_1}{2}\cdot\frac{1}{2\cdot3\ldots2n-1} + \frac{\mathrm{B}_3}{2\cdot3\cdot4}\cdot\frac{1}{2\cdot3\ldots2n-3} + \\ + \frac{\mathrm{B}_5}{2\cdot3\cdot4\cdot5\cdot6}\cdot\frac{1}{2\cdot3\ldots2n-5} + \cdots + \frac{\mathrm{B}_{2n-1}}{2\cdot3\ldots2n}\cdot1 = 0 \end{array}\right\} \quad (7.)$$

Multiplying every term by (2·3…2*n*) we have

$$\left.\begin{array}{l} 0 = -\frac{1}{2}\cdot\frac{2n-1}{2n+1} + \mathrm{B}_1\left(\frac{2n}{2}\right) + \mathrm{B}_3\left(\frac{2n\cdot(2n-1)\cdot(2n-2)}{2\cdot3\cdot4}\right) + \\ + \mathrm{B}_5\left(\frac{2n\cdot(2n-1)\ldots(2n-4)}{2\cdot3\cdot4\cdot5\cdot6}\right) + \cdots + \mathrm{B}_{2n-1} \end{array}\right\} \quad (8.)$$

which it may be convenient to write under the general form:

$$0 = \mathrm{A}_0 + \mathrm{A}_1\mathrm{B}_1 + \mathrm{A}_3\mathrm{B}_3 + \mathrm{A}_5\mathrm{B}_5 + \cdots + \mathrm{B}_{2n-1} \quad (9.)$$

A_1, A_3, &c. being those functions of *n* which respectively belong to B_1, B_3, &c.

We might have derived a form nearly similar to (8.), from D_{2n-1} the coefficient of any *odd* power of *x* in (6.); but the general form is a little different for the coefficients of the *odd* powers, and not quite so convenient.

On examining (7.) and (8.), we perceive that, when these formulæ are isolated from (6.), whence they are derived, and considered in themselves separately and independently, *n* may be any whole number whatever; although when (7.) occurs as *one of the* D's in (6.), it is obvious that *n* is then not arbitrary, but is always a certain function of the *distance of that* D *from the beginning*. If that distance be =*d*, then

$$\begin{array}{c} 2n + 1 = d, \text{ and } n = \frac{d-1}{2} \text{ (for any } \textit{even} \text{ power of } x) \\ 2n = d, \text{ and } n = \frac{d}{2} \text{ (for any } \textit{odd} \text{ power of } x). \end{array}$$

It is with the *independent* formula (8.) that we have to do. Therefore it must be remembered that the conditions for the value of *n* are now modified, and that *n* is a perfectly *arbitrary* whole number. This circumstance, combined with the fact (which we may easily perceive) that whatever *n* is, every term of (8.) after the (*n*+1)th is =0, and that the (*n*+1)th term itself is always $= \mathrm{B}_{2n-1}\cdot\frac{1}{1} = \mathrm{B}_{2n-1}$, enables us to find the value (either numerical or algebraical) of any *n*th Number of Bernoulli B_{2n-1}, *in terms of all the preceding ones*, if we but know the values of B_1, B_3…B_{2n-3}. We append to this Note a Diagram and Table, containing the details of the computation for B_7 (B_1, B_3, B_5 being supposed given).

On attentively considering (8.), we shall likewise perceive that we may derive from it the numerical value of *every* Number of Bernoulli in succession, from the very beginning, *ad infinitum*, by the following series of computations:

- 1st Series.—Let *n*=1, and calculate (8.) for this value of *n*. The result is B_1.
- 2nd Series.—Let *n*=2. Calculate (8.) for this value of *n*, substituting the value of B_1 just obtained. The result is B_3.
- 3rd Series.—Let *n*=3. Calculate (8.) for this value of *n*, substituting the values of B_1, B_3 before obtained. The result is B_5. And so on, to any extent.

The diagram[29] represents the columns of the engine when just prepared for computing B_{2n-1} (in the case of *n*=4); while the table beneath them presents a complete simultaneous view of all the successive changes which these columns then severally pass through in order to perform the computation. (The reader is referred to Note D. for explanations respecting the nature and notation of such tables.)

Six numerical *data* are in this case necessary for making the requisite combinations. These data are 1, 2, *n*(=4), B_1, B_3, B_5. Were *n*=5, the additional datum B7 would be needed. Were *n*=6, the datum B9 would be needed; and so on. Thus the actual *number of data* needed will always be *n*+2, for *n*=*n*; and out of these *n*+2 data, $(n+2-3)$ of them are successive Numbers of Bernoulli. The reason why the Bernoulli Numbers used as data are nevertheless placed on *Result*-columns in the diagram, is because they may properly be supposed to have been previously computed in succession by the *engine* itself; under which circumstances each B will appear as a *result*, previous to being used as a *datum* for computing the succeeding B. Here then is an instance (of the kind alluded to in Note D.) of the same Variables filling more

[29] The diagram of the computation of the Numbers of Bernoulli is very large, so it is shown in a separate page at the end of paper.

than one office in turn. It is true that if we consider our computation of B_7 as a perfectly isolated calculation, we may conclude B_1, B_3, B_5 to have been arbitrarily placed on the columns; and it would then perhaps be more consistent to put them on V_4, V_5, V_6 as data and not results. But we are not taking this view. On the contrary, we suppose the engine to be *in the course of* computing the Numbers to an indefinite extent, from the very beginning; and that we merely single out, by way of example, *one amongst* the successive but distinct series of computations it is thus performing. Where the B's are fractional, it must be understood that they are computed and appear in the notation of *decimal* fractions. Indeed this is a circumstance that should be noticed with reference to all calculations. In any of the examples already given in the translation and in the Notes, some of the *data*, or of the temporary or permanent results, might be fractional, quite as probably as whole numbers. But the arrangements are so made, that the nature of the processes would be the same as for whole numbers.

In the above table and diagram we are not considering the *signs* of any of the B's, merely their numerical magnitude. The engine would bring out the sign for each of them correctly of course, but we cannot enter on *every* additional detail of this kind as we might wish to do. The circles for the signs are therefore intentionally left blank in the diagram.

Operation-cards 1, 2, 3, 4, 5, 6 prepare $-\frac{1}{2} \cdot \frac{2n-1}{2n+1}$. Thus, Card 1 multiplies *two* into *n*, and the three *Receiving* Variable-cards belonging respectively to V4, V5, V6, allow the result 2*n* to be placed on each of these latter columns (this being a case in which a triple receipt of the result is needed for subsequent purposes); we see that the upper indices of the two Variables used, during Operation 1, remain unaltered.

We shall not go through the details of every operation singly, since the table and diagram sufficiently indicate them; we shall merely notice some few peculiar cases.

By Operation 6, a *positive* quantity is turned into a *negative* quantity, by simply subtracting the quantity from a column which has only zero upon it. (The sign at the top of V8 would become − during this process.)

Operation 7 will be unintelligible, unless it be remembered that if we were calculating for *n* = 1 instead of n = 4, Operation 6 would have completed the computation of B1 itself, in which case the engine instead of continuing its processes, would have to put B_1 on V_{21}; and then either to stop altogether, or to begin Operations 1, 2…7 all over again for value of *n*(=2), in order to enter on the computation of B_3; (having however taken care, previous to this recommencement, to make the number on V_3 equal to *two*, by the addition of unity to the former *n*=1 on that column). Now Operation 7 must either bring out a result equal to zero (if *n*=1); or a result *greater* than *zero*, as in the present case; and the engine follows the one or the other of the two courses just explained, contingently on the one or the other result of Operation 7. In order fully to perceive the necessity of this *experimental* operation, it is important to keep in mind what was pointed out, that we are not treating a perfectly isolated and independent computation, but one out of a series of antecedent and prospective computations.

Cards 8, 9, 10 produce $-\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \frac{2n}{2}$. In Operation 9 we see an example of an upper index which again becomes a value after having passed from preceding values to zero. V_{11} has successively been ${}^0V_{11}$, ${}^1V_{11}$, ${}^2V_{11}$, ${}^0V_{11}$, ${}^3V_{11}$; and, from the nature of the office which V_{11} performs in the calculation, its index will continue to go through further changes of the same description, which, if examined, will be found to be regular and periodic.

Card 12 has to perform the same office as Card 7 did in the preceding section; since, if *n* had been =2, the 11th operation would have completed the computation of B_3.

Cards 13 to 20 make A_3. Since A_{2n-1} always consists of 2*n*-1 factors, A_3 has three factors; and it will be seen that Cards 13, 14, 15, 16 make the second of these factors, and then multiply it with the first; and that 17, 18, 19, 20 make the third factor, and then multiply this with the product of the two former factors.

Card 23 has the office of Cards 11 and 7 to perform, since if *n* were =3, the 21st and 22nd operations would complete the computation of B_5. As our case is B_7, the computation will continue one more stage; and we must now direct attention to the fact, that in order to compute A_7 it is merely necessary precisely to repeat the group of Operations 13 to 20; and then, in order to complete the computation of B_7, to repeat Operations 21, 22.

It will be perceived that every unit added to *n* in B_{2n-1}, entails an additional repetition of operations (13…23) for the computation of B_{2n-1}. Not only are all the *operations* precisely the same however for every such repetition, but they require to be respectively supplied with numbers from the very *same pairs of columns*; with only the one exception of Operation 21, which will of course need B_5 (from V_{23}) instead of B_3 (from V_{22}). This identity in the *columns* which supply the requisite numbers must not be confounded with identity in the *values* those columns have upon them and give out to the mill. Most of those values undergo alterations during a performance of the operations (13…23), and consequently the columns present a new set of values for the *next* performance of (13…23) to work on.

At the termination of the *repetition* of operations (13…23) in computing B_7, the alterations in the values on the Variables are, that

V_6 = 2*n*-4 instead of 2*n*-2.
V_7 = 6 . . . . . 4.
V_{10} = 0 . . . . . 1.
V_{13} = $A_0+A_1B_1+A_3B_3+A_5B_5$ instead of $A_0+A_1B_1+A_3B_3$.

In this state the only remaining processes are, first, to transfer the value which is on V_{13} to V_{24}; and secondly, to reduce V_6, V_7, V_{13} to zero, and to add one[30] to V_3, in order that the engine may be ready to commence computing B_9. Operations 24 and 25 accomplish these purposes. It may be thought anomalous that

[30] It is interesting to observe, that so complicated a case as this calculation of the Bernoullian Numbers nevertheless presents a remarkable simplicity in one respect; viz. that during the processes for the computation of *millions* of these Numbers, no other arbitrary modification would be requisite in the arrangements, excepting the above simple and uniform provision for causing one of the data periodically to receive the finite increment unity.

Operation 25 is represented as leaving the upper index of V_3 still=unity; but it must be remembered that these indices always begin anew for a separate calculation, and that Operation 25 places upon V_3 the *first* value *for the new calculation*.

It should be remarked, that when the group (13…23) is *repeated*, changes occur in some of the *upper* indices during the course of the repetition: for example, ${}^3V_6$ would become ${}^4V_6$ and ${}^5V_6$.

We thus see that when *n*=1, nine Operation-cards are used; that when *n*=2, fourteen Operation-cards are used; and that when *n*>2, twenty-five Operation-cards are used; but that no *more* are needed, however great *n* may be; and not only this, but that these same twenty-five cards suffice for the successive computation of all the Numbers from B_1 to B_{2n-1} inclusive. With respect to the number of *Variable*-cards, it will be remembered, from the explanations in previous Notes, that an average of three such cards to each *operation* (not however to each Operation-*card*) is the estimate. According to this, the computation of B_1 will require twenty-seven Variable-cards; B_3 forty-two such cards; B_5 seventy-five; and for every succeeding B after B_5, there would be thirty-three additional Variable-cards (since each repetition of the group (13…23) adds eleven to the number of operations required for computing the previous B). But we must now explain, that whenever there is a *cycle of operations*, and if these merely require to be supplied with numbers from the *same pairs of columns*, and likewise each operation to place its *result* on the *same* column for every repetition of the whole group, the process then admits of a *cycle of Variable-cards* for effecting its purposes. There is obviously much more symmetry and simplicity in the arrangements, when cases do admit of repeating the Variable as well as the Operation-cards. Our present example is of this nature. The only exception to a *perfect identity* in *all* the processes and columns used, for every repetition of Operations (13…23), is, that Operation 21 always requires one of its factors from a new column, and Operation 24 always puts its result on a new column. But as these variations follow the same law at each repetition (Operation 21 always requiring its factor from a column *one* in advance of that which it used the previous time, and Operation 24 always putting its result on the column *one* in advance of that which received the previous result), they are easily provided for in arranging the recurring group (or cycle) of Variable-cards.

We may here remark, that the average estimate of three Variable-cards coming into use to each operation, is not to be taken as an absolutely and literally correct amount for all cases and circumstances. Many special circumstances, either in the nature of a problem, or in the arrangements of the engine under certain contingencies, influence and modify this average to a greater or less extent; but it is a very safe and correct *general* rule to go upon. In the preceding case it will give us seventy-five Variable-cards as the total number which will be necessary for computing any B after B_3. This is very nearly the precise amount really used, but we cannot here enter into the minutiæ of the few particular circumstances which occur in this example (as indeed at some one stage or other of probably most computations) to modify slightly this number.

It will be obvious that the very *same* seventy-five Variable-cards may be repeated for the computation of every succeeding Number, just on the same principle as admits of the repetition of the thirty-three Variable-cards of Operations (13…23) in the computation of any *one* Number. Thus there will be a *cycle of a cycle* of Variable-cards.

If we now apply the notation for cycles, as explained in Note E., we may express the operations for computing the Numbers of Bernoulli in the following manner:

(1…7), (24, 25) ……………………	gives B_1	=1st number;	(n being = 1).
(1…7), (8…12), (24, 25) …………………	B_3	=2nd ……;	(n …… = 2).
(1…7), (8…12), (13…23), (24, 25) …………	B_5	=3rd ……;	(n …… = 3).
(1…7), (8…12), 2(13…23), (24, 25) …………	B_7	=4th ……;	(n …… = 4).
………………………………			
………………………………			
(1…7), (8…12), $\Sigma(+1)^{n-2}$(13…23), (24, 25)…	B_{2n-1}	=nth ……;	(n …… = n).

Again,

$$\underset{\text{limits 1 to } n}{(1\ldots7), (24,25), \Sigma(+1)^n} \left\{ \underset{\text{limits 0 to } (n+2)}{(1\ldots7), (8\ldots12), \Sigma(n+2)(13\ldots23), (24,25)} \right\}$$

represents the total operations for computing every number in succession, from B_1 to B_{2n-1} inclusive.

In this formula we see a *varying cycle* of the *first* order, and an ordinary cycle of the *second* order. The latter cycle in this case includes in it the varying cycle.

On inspecting the ten Working-Variables of the diagram, it will be perceived, that although the *value* on any one of them (excepting V_4 and V_5) goes through a series of changes, the *office* which each performs is in this calculation *fixed* and *invariable*. Thus V_6 always prepares the *numerators* of the factors of any A; V_7 the *denominators*. V_8 always receives the (2*n*-3)th factor of A_{2n-1}, and V_9 the (2*n*-1)th. V_{10} always decides which of two courses the succeeding processes are to follow, by feeling for the value of *n* through means of a subtraction; and so on; but we shall not enumerate further. It is desirable in all calculations so to arrange the processes, that the *offices* performed by the Variables may be as uniform and fixed as possible.

Supposing that it was desired not only to tabulate B_1, B_3, &c., but A_0, A_1, &c.; we have only then to appoint another series of Variables, V_{41}, V_{42}, &c., for receiving these latter results as they are successively produced upon V_{11}. Or again, we may, instead of this, or in addition to this second series of results, wish to tabulate the value of each successive *total* term of the series (8.), viz. A_0, A_1B_1, A_3B_3, &c. We have then merely to multiply each B with each corresponding A, as produced, and to place these successive products on Result-columns appointed for the purpose.

The formula (8.) is interesting in another point of view. It is one particular case of the general Integral of the following Equation of Mixed Differences:

$$\frac{d^2}{dx^2}\left(z_{n+1}x^{2n+2}\right) = (2n+1)(2n+2)z^n x^{2n}$$

for certain special suppositions respecting *z*, *x* and *n*.

The *general* integral itself is of the form,

$$z_n = f(n)\cdot x + f_1(n) + f_2(n)\cdot x^{-1} + f_3(n)\cdot x^{-3} + \cdots$$

and it is worthy of remark, that the engine might (in a manner more or less similar to the preceding) calculate the value of this formula upon most *other* hypotheses for the functions in the integral with as much, or (in many cases) with more ease than it can formula (8.).

A. A. L.

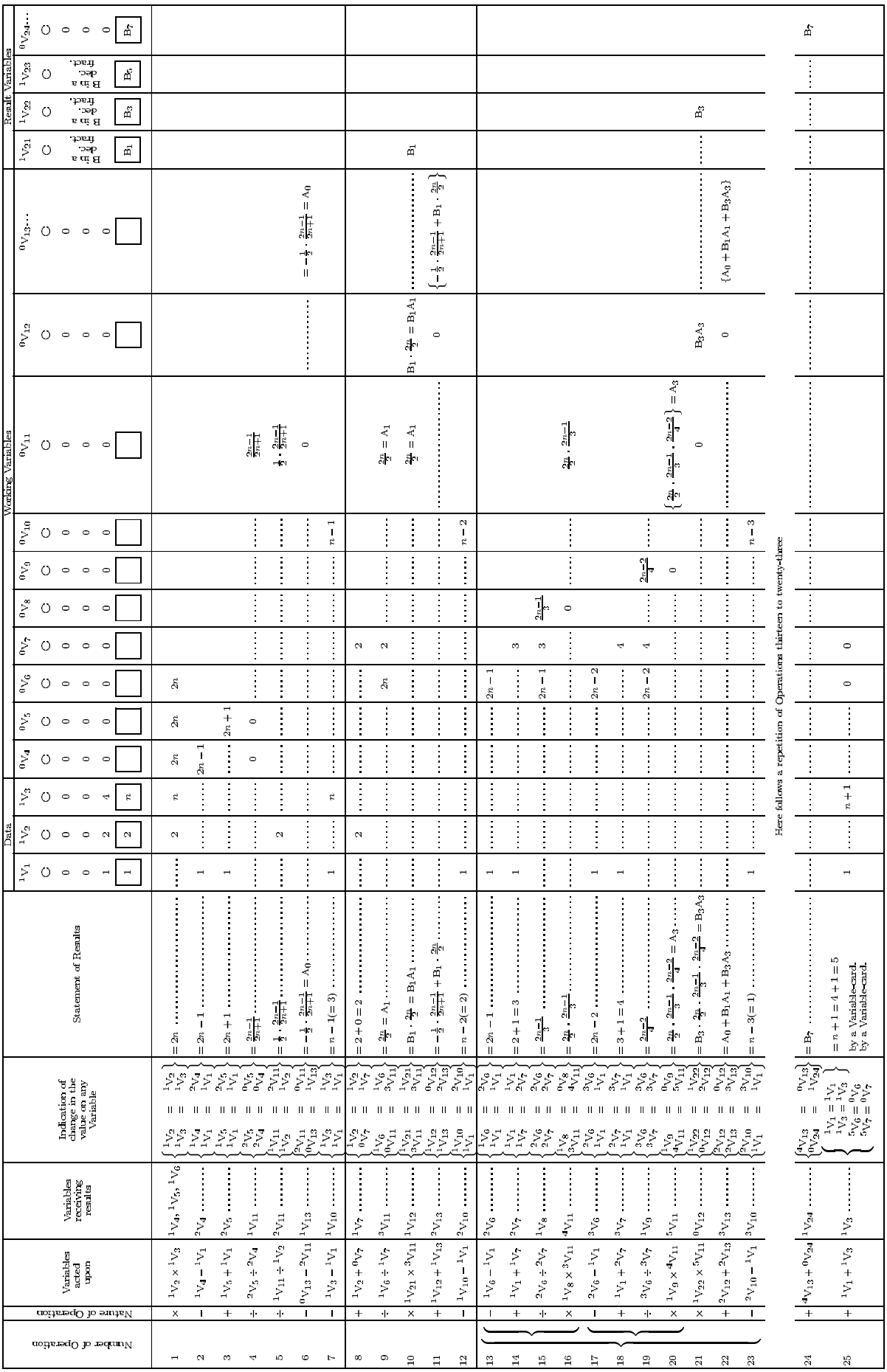


Figure 1: Diagram of the computation of the Numbers of Bernoulli

Effective Worst-Case Execution Time Analysis of DO178C Level A Software

Stephen Law, Mike Bennett, Stuart Hutchesson, Ivan Ellis
Rolls-Royce Controls & Data Services Ltd, Birmingham, UK; email: {stephen.law|mike.bennett|stuart.hutchesson|ivan.ellis}@controlsdata.com
Guillem Bernat, Antoine Colin, Andrew Coombes
Rapita Systems Ltd, Atlas House, Osbaldwick Link Road, York; email: {bernat|colin|acoombes}@rapitasystems.com

Abstract

This paper presents the industrial experience of applying an approach to Worst-Case Execution Time (WCET) Analysis that combines analysis and measurement to support a newly developed embedded microprocessor. It outlines an effective method that provides robust results to support certification requirements and offers additional business advantages. The approach outlined in this paper is shown to have been made possible through the use of a highly-structured software architecture, reuse of existing test cases, careful hardware design and use of a Commercial-Off-The-Shelf (COTS) toolset: RapiTime provided by Rapita Systems Ltd.

1 Introduction

Determining the WCET of a program is a key component for demonstrating correct operation of software; specifically as input to schedulability analysis. Correct timing performance is a critical characteristic of engine control systems and problems with timing can be the most difficult and costly to find and fix. A safe, but not overly pessimistic, WCET is fundamental to understanding the system performance; getting it wrong may result in a misunderstanding of overall system behaviour. DO178C/ED-12C [1] recognises that the guidance provided in DO178B/ED-12B [2] no longer reflects the complexities of modern hardware and software engineering. It states that a combination of reviews, analysis and testing may be needed to provide adequate verification of the WCET of a piece of software.

Rolls Royce Controls and Data Services (CDS) develops DO178B/C Level A compliant software for a variety of aircraft engine control applications with critical timing performance requirements. Key to this process is the use of a time analysable processor developed by CDS. The **VISIUM**CORE™ is the newest iteration of the CDS DO-254 Level A obsolescence-protected microprocessor which is designed specifically for operation in harsh-environments. Rapita Systems Ltd. provides solutions for software verification and more specifically, Worst-Case Execution Time (WCET) Analysis.

This paper discusses an approach to WCET analysis that has been applied to the CDS **VISIUM**CORE™ processor. The process builds upon an existing low level component testing setup which obtains full MC/DC (Modified Condition/Decision Coverage) coverage through the system under test by following a compartmentalised test strategy. The WCET of individual components is computed by RapiTime separately for each component and the overall WCET of the full program is computed analytically by RapiTime by "rolling-up" the timings of the individual components. The paper argues that the proposed technique meets certification requirements and offers advantages over alternate approaches. Section 1.1 provides a short recap of the WCET problem. Section 1.2 provides an overview of typical techniques used in industry to solve the WCET problem. Section 2 introduces the new CDS approach, finally, section 3 describes the results of the effectiveness of RapiTime and the comparison with the previous static analysis approach.

1.1 The Worst Case Execution Time Problem

Practical WCET analysis techniques can be thought of attempting to solve two fundamental problems (see [3] for a detailed survey of WCET methods and techniques).

The first is finding the worst-case path through the program structure, or graph. This is non-trivial unless the software architecture and design are constrained. For example, this could be through the bounding of loops, or prevention of recursion. Even if this is the case, there is often a complex mapping between the various levels of program representation (ie. model-level, source-code level or object-code level). For example, compilers can apply complex optimizations that create loop constructs to represent straight-line code. This means that even if the engineer understands the worst-case path in a high-level representation, this may not translate well into an understanding of the object-code that executes on a real computing platform.

The second problem is determining the length of time a path of this software takes to execute on a real piece of computing hardware. This is also non-trivial in modern, complex hardware due to the consequences of processor features such as out of order execution, or caches (particularly with respect to timing anomalies [4]).

There are three key methods of WCET analysis: static analysis, pure measurement and hybrid methods.

Static analysis takes the source code or object code of the System Under Test (SUT), analyses the possible paths through the code, and by modelling the target hardware; calculates which path through the SUT will produce the WCET. The analysis is wholly reliant on the underlying model of the target hardware, but gains from being able to fully examine the full set of paths through the SUT. One of the primary drawbacks of static analysis techniques is the technique's reliance on accurate processor models, as developers look to use ever more complex processors the complexity of these models increases accordingly, a secondary drawback is the lack of information in the code to determine accurate loop bounds and context dependent information. It has even been suggested that static analysis techniques has already 'hit the complexity wall of today's processors' [5].

A large proportion of industry has followed traditionally a "pure" measurement approach where the SUT is measured end-to-end (see High-watermark below). Measurements have the advantage of capturing what is really happening at the processor level without having to make any assumption about its behaviour, however, the main drawback is that there is, in general, no guarantee that the worst path, or state within the worst-path has been exercised.

The hybrid approach (used by RapiTime) combines the benefits of both approaches. This is based on using measurements of smalls sections of code, but also using structural static path analysis techniques to combine the measurements of these small sections of code into an overall WCET. An additional advantage is that as part of the measurement process a comparison of the differences between end-to-end measurements and static analysis can be performed. This provides further evidence and confidence on how well tests drive the worst-case path. Furthermore these time measurements can be derived from the actual target hardware, with no reliance on complex timing models. However the technique suffers from the fact that the software must be executed on the target hardware to a sufficient level to provide accurate results.

1.2 Typical Industrial Solutions

Typical approaches used within the aerospace industry use one or more of the following techniques:

- **High-water marking** An end-to-end measurement on the target processor is recorded when it exceeds a previously recorded value, for a given piece of software. The benefit of this method is that it is simple to implement, accurate and scales well to large software. It can be implemented on any microprocessor and can capture results during all levels of end-to-end testing. However, the disadvantage is that it is not possible, in general, to identify the worst-case test vector that would lead to the worst-case execution path and therefore can lead to optimistic estimates unless it is combined with robust (typically manual) analysis to show that the worst-case path has been executed, if indeed such analysis is possible.
- **Code Structure Analysis** This technique supports the alternative approach of analysis through extracting the longest path from the source or object-code representation of the program. Typically this is supported by tooling (but in some instances, only a manual approach is possible) and may rely on annotating the code to provide more information (eg. maximum loop counts). Object-code analysis potentially provides more accurate analysis, compared to source-code analysis as it is able to cater for the complex optimisations that may be introduced during the compilation process. However, it is less portable and may be more difficult for engineers to apply.
- **Processor Modelling or Simulation** To obtain low-level timings of software, a model or simulation of the processor can be built that abstracts the operation of the processor sufficiently, to make analysis tractable, whilst giving providing results that are neither optimistic (ie. invalid) or too pessimistic (and therefore not useful). As already discussed, developing and verifying a model of a processor can be a complex and time-consuming task. This was the basis of the technique used within CDS prior to the introduction of the latest generation of **VISIUM**CORE™ processor.
- **Incorporation of Safety Margin** Because it is difficult to show that both modelling and measurement are not optimistic, an additional *safety margin* may be added to the calculated worst-case, or built into the overall processor loading, that ensures that the processor remains lightly loaded. This has the twin-issues of wastefulness and potential inadequacy.

1.3 Rapita Systems RapiTime

RapiTime [1] is a tool developed by Rapita Systems for performing timing analysis (including analytical WCET and High-watermark) based on the hybrid static/measurement approach. The tool automatically instruments Ada, C and C++ and automatically analyses the structure of the code. When the instrumented code runs on the target the instrumentation produces a timing trace that is then analysed off-line and together with a high-level structural analysis, WCET and other timing analysis reports are produced and presented via a GUI. The RapiTime tool is highly configurable and supports a whole range of WCET analysis strategies.

2 Effective Worst-Case Execution Time Analysis

2.1 Overview

CDS are currently developing an improved process to support the **VISIUM**CORE™ processor based on a combination of measurement on target and source-code structure analysis. The remainder of this section briefly introduces this and details the reasons why this is a valid approach. The approach is summarised in Figure 2 which shows a simplified view of the process.

[1] https://www.rapitasystems.com/products/rapitime

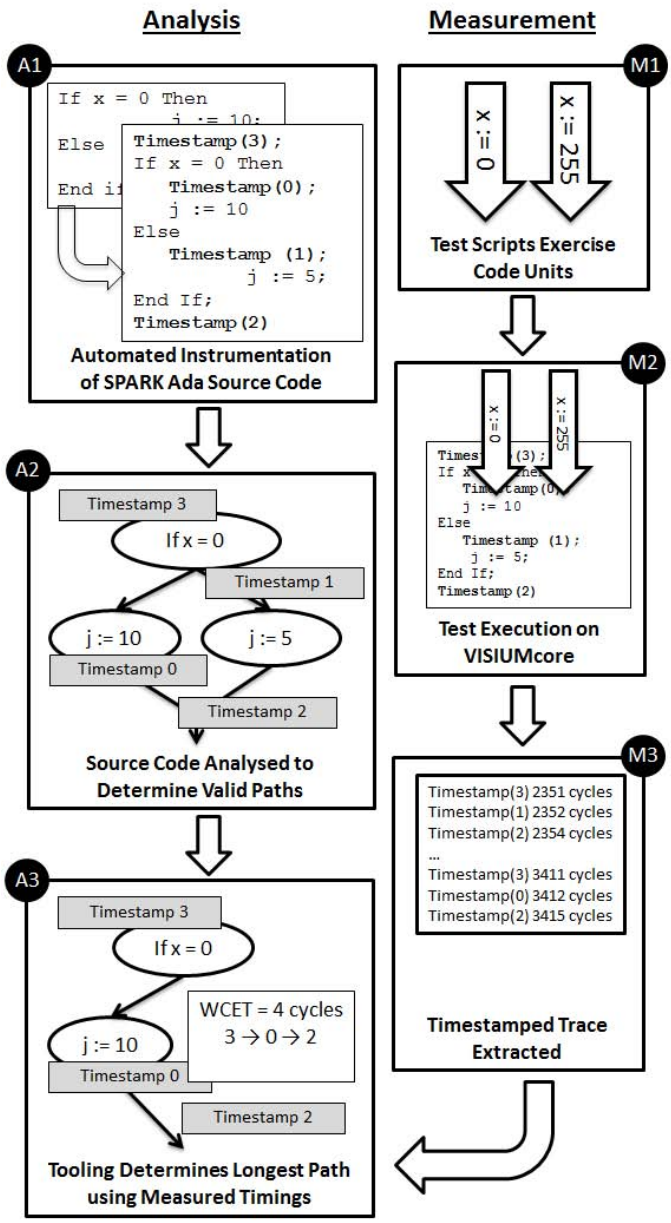


Figure 1: Overview of WCET Analysis Process

In A1 the SPARK Ada 95 source code is instrumented automatically with timestamps that allows all paths in the compiler provided Control Flow Graph (CFG) to be measured individually. This instrumented software is compiled within a harness that allows units of the software to be tested in isolation on the target processor. In steps M1 and M2 unit test scripts exercise the code to achieve full coverage of the timestamps on the **VISIUM**CORE™ processor. The download and execution process is fully automated and unit test results are able to be captured at the same time. In step M3, the timestamped trace is captured using a high-speed logic analyser and processed into a form that can be read by the RapiTime toolset.

To complete the WCET calculation, step A2 analyses the program graph to determine the feasible paths. The toolset allows annotations to remove infeasible paths, if analysis is unduly pessimistic. Step A3 then uses the timestamp measurements to calculate the WCET of each module. In order to combine the unit timings, the toolset aggregates the timings to provide an overall end-to-end thread timing that is the primary input for schedulability analysis. The advantage of this approach is that WCET analysis is, in effect, performed as a side effect of the module testing without the need to perform any system level testing or any timing specific testing.

2.2 Detail

The process adopted by CDS has been made possible by a number of factors. Some key ones are discussed below:

A software architecture and philosophy amenable to analysis

The CDS engine control software architecture, based on [6], is a layered architecture that provides standardized interfaces between the hardware and the various application layers. Functions are scheduled as threads that form the architecture's schedulable entities. These threads follow a well-defined calling sequence. The scheduler used is fixed-priority, non-preemptive. This means that WCET analysis need not consider variation due to pre-emption.

A processing architecture that supports analysis and provides accurate non-intrusive tracing and time-stamping of software execution

The **VISIUM**CORE™ is a packaged device that integrates a core, memory, IO and tracepoint interfaces. Being targeted at the safety-critical embedded sector, the device is DO-254 – Level A compliant. It has extensive Single-Event-Effect (SEE) protection and is suitable for harsh environments.

The design of the core balances performance and support for easing software certification. The latest version of the **VISIUM**CORE™ features a five-stage superscaler pipeline, with multiple execution units allowing managed parallel execution. The processor also implements simple static block prediction logic. The processor does not incorporate a data or instruction cache, and carefully managed memory devices to remove the risk of memory bus contention.

The processor has been carefully designed to ensure that each instruction's execution is time-invariant; in other words each instruction will take the same time to execute, regardless of the data its operation is performed upon. These design features ensure that previous processor state has no effect on the current operation of the device.

To enable timing of functions, the processor provides facilities to non-intrusively collect an entire instruction trace complete with timestamps. The **VISIUM**CORE™ has also been augmented with functionality to output a user-specified value and timestamp. Both the trace facility and the instruction are low-overhead, incurring only a single instruction fetch.

The trace facility is an independent component within the processor, separate to all peripherals. The output of tracepoints is performed on a reserved interface, thus allowing tracepoints to be safely, non-intrusively, kept in the final software verified and delivered with no disturbance on data buses.

Extensive Coverage provided by low-level test schedules

CDS utilise standard low-level test (LLT) tools to provide the Modified Condition Decision Coverage (MC/DC) required to meet DO178B/C objectives. Test schedules exist that are able to provide coverage of the majority of the software as part of the normal verification process. Where coverage is provided through other verification vehicles, schedules must be created to meet the coverage requirements, but they need not provide MCDC coverage.

The use of LLT, or unit, testing allows detailed coverage to be obtained at a small level, on a single processor. The technique allows code to be verified and analysed at a time in the design life-cycle when full system testing, on system hardware is not yet possible. The tool used to facilitate this verification allows lower level functions to be replaced with test code, allowing the function under test to be easily manipulated. These replaced functions are then tested using additional component tests.

Each loop through the code is either tested to its maximum number of iterations by each test script, or is annotated by engineers at design time.

Comparison between code under test and code in the final system

It is already established practice within CDS to automatically compare the object code generated for the component under test against the object code in the final executable image. This provides evidence that both the functional and timing tests are representative. The features of the software and hardware designs mean that the context in which the executable runs does not invalidate either the functional or timing tests. For example, the test may execute at a different location in memory, but this has no effect on the timing.

Optimized queuing, running and collation of results from target hardware

To facilitate timing the large amount of software within an Electronic Engine Controller (EEC) or similar application, a large number of test schedules must be executed. In each case, the test must be uploaded to target, programmed in non-volatile memory, executed and results downloaded. To facilitate this, a large number of identical hardware units have been built within a CDS designed test rack. The high-speed logic analyser provided by Rapita Systems is used to collect results.

Combining unit tests to provide a system level result

The final step in the analysis process is to combine the results obtained through component testing, to provide a system level result. To perform this a new RapiTime add-on tool was developed. The tool analyses a full set of results files and automatically extracts timing information for each function in the system under test to produce a combined system level result. The ability to produce a credible system level result in this way is facilitated thanks to the architecture of the **VISIUM**CORE™; as the previous processor state at entry point to each function can be ignored.

The tool also provides the capability for regression testing, where some components within a result can be replaced with updated code, and results. The ability to manage the combining of small components to form a system level result is provided through robust configuration management which ensures that only the applicable results for the current system are utulised.

A process and toolset tailored to CDS requirements

Rapita Systems and CDS have worked closely to develop, implement and validate the WCET process as part of a multi-year project. Rapita Systems have supported the target integration with the **VISIUM**CORE™ processor through on-site and remote working with CDS equipment. The relationship has been mutually beneficial, it has driven enhancements in the Rapita Systems toolset to offer new capabilities. It has also driven improvements to CDS coding standards to further enhance the analysability of the code under test.

3 Status and Initial Results

In order to assess the suitability of RapiTime to CDS tools and processes, an initial study was conducted on a previous version of the CDS processor, this previous version of the processor uses an in-house static analysis tool to perform WCET analysis. This tool is built upon a cycle accurate simulator and uses path analysis to identify a safe WCET figure.

RapiTime was applied to this previous CDS processor, the results obtained were then compared to the results obtained through the use of the CDS static analysis tool, and timings obtained through High-Water-Mark (HWM) testing of the SUT.

The results obtained are shown in Table 1, where LOC represents the number of lines of code in the test code item. The static, HWM and RapiTime results illustrate the results produced through static analysis, test HWM figures, and RapiTime calculated figures.

All RapiTime results provided WCET values higher than the HWM observed through testing. An in depth analysis was conducted into each item where the static time differed from the RapiTime WCET. The majority of cases were due to pessimistic loop count estimations on the part of the Static tool, for instance this was observed with DP, DTC and DT. In other cases the difference was due to pessimistic hardware memory access times, this was particularly prevalent with the I function, this pessimism was exacerbated due to the accesses location within a loop. In the case of DE the static analysis tool was proved to be pessimistic due to its inability to take into account the calling context of a large child-function.

Following this initial analysis a full scale trial was conducted using RapiTime targeting the **VISIUM**CORE™ processor. The tool was used to analyse two engine control system builds

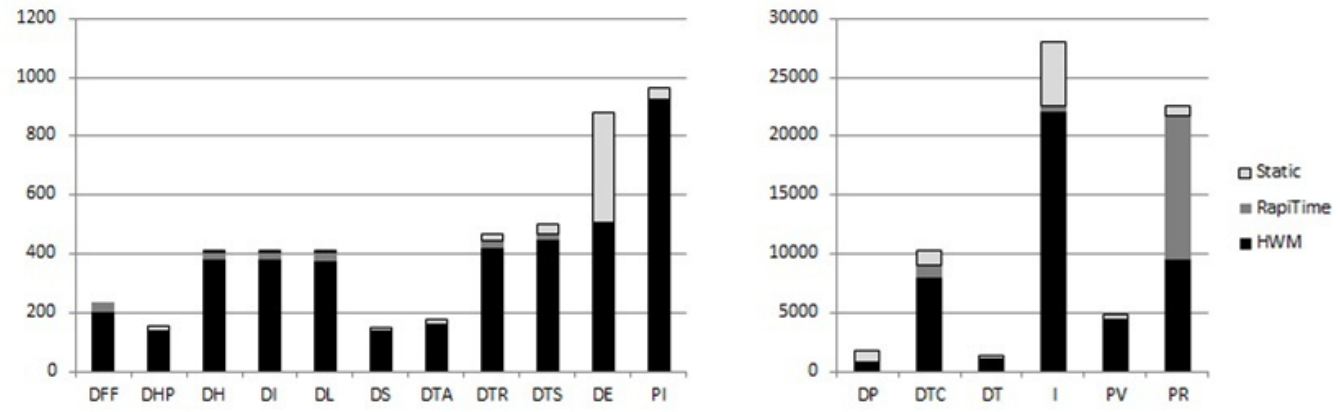


Figure 2: Comparison Between Static, HWM and RapiTime WCET Figures for 25 Engine Controller Functions

Table 1

Test Item	LOC	Static	HWM	RapiTime
DFF	85	234	203	236
DHP	22	154	140	140
DH	167	409	379	403
DI	164	409	379	403
DL	164	409	375	406
DP	254	1740	702	731
DS	78	146	138	138
DTC	210	10337	7906	9045
DTA	27	174	160	160
DTR	180	466	419	446
DTS	189	501	449	467
DT	395	1356	985	1104
DE	108	882	506	508
I	578	28006	22056	22511
PI	33	966	926	926
PV	105	4794	4402	4454
PR	264	22580	9511	21774

targeting different production units. In total over 8000 component test scripts were executed, with their results being compared to HWM results obtained during testing. Additions, or modifications, were required in less than 5% of test scripts, and less than 1% of formal software components.

All 8000 tests produced results larger than the HWM results observed during testing. The results were subsequently rolled up to produce a system wide WCET, in total this encompassed an analysis of over 400,000 blocks of code.

The tool has been integrated into CDS' software build and verification system, which utilises AdaCore GNATPro 7.2.7 [2], and has been qualified according to DO-178C as a TQL5 verification tool.

4 Conclusion

This paper introduced the application of a hybrid, measurement based WCET analysis tool to an industrial safety-critical system. An extensive study was performed on the tool to assess its feasibility and suitability to the application. This study included an assessment of the tool's design and its integration with the target processor architecture & target software architecture. This study was conducted using real engine control application software. The WCET figures produced by the tool have been verified against results calculated by a legacy, qualified static analysis tool, and against end-to-end timing measurements obtained through extensive testing. The study proved the validity of the technique, and justified the tool's use in a DO-178B/C environment.

The tool is now being rolled out across multiple sites for use by all CDS projects utilising the **VISIUM**CORE™ processor.

[2] http://www.adacore.com/gnatpro

References

[1] RTCA (2011), *DO-178C - Software Considerations in Airborne Systems and Equipment Certification*.

[2] RTCA (1992), *DO-178B - Software Considerations in Airborne Systems and Equipment Certification*.

[3] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström (2008), *The worst-case execution-time problem—overview of methods and survey of tools*, ACM Transactions on Embedded Computing Systems, vol. 7, no. 3, p. 38.

[4] T. Lundqvist and P. Stenstrom (1999), *Timing anomalies in dynamically scheduled microprocessors*, in Proceedings of The 20th Real-Time Systems Symposium, pp. 12–21, IEEE.

[5] R. Kirner and P. Puschner (2008), *Obstacles in worst-case execution time analysis*, in Proceedings of the 11th International Symposium on Object Oriented Real-Time Distributed Computing, pp. 333–339, IEEE.

[6] S. Hutchesson (2013), *Trusted Product Lines*. PhD thesis.

The Need for a Weaving Model in Assurance Case Automation

R. Hawkins, I. Habli, T. Kelly

The Department of Computer Science, The University of York, York, YO10 5GH. Tel: +44 1904 325463; email: {richard.hawkins|ibrahim.habli|tim.kelly}@york.ac.uk

Abstract

*In this paper we describe how the automated instantiation of assurance case arguments will require information to be extracted from multiple models of a system and its environment and engineering processes, e.g. safety and verification processes. For this to be done successfully the dependencies between the models must be explicitly, completely and correctly captured. We describe how a model-based approach, **model weaving**, provides an excellent mechanism for modelling the correspondences that exist between models and discuss how model weaving can be applied in the context of assurance cases.*

1 Introduction

Assurance cases provide an explicit means for justifying and assessing confidence in critical properties of interest such as safety or security properties. An assurance case should contain a reasoned and compelling argument, supported by a body of evidence [1]. We are concerned with the challenge of how to make it easier for system developers to create valid and compelling arguments for their systems. To help to guide assurance argument development, the concept of providing reusable patterns of argument and evidence was developed [2]. Assurance argument patterns allow the desired structure of the argument to be captured, whilst abstracting from the details of a particular target system. An assurance argument can be created for a system by instantiating the argument pattern with information about the target system. Assurance argument patterns have been shown to be useful in helping developers create arguments [3]. However current practice is mainly to instantiate argument patterns manually.

There are a number of advantages to be gained from automating the generation of assurance arguments:

- Human error in instantiating patterns is eliminated.
- The argument can be generated directly from, and is therefore consistent with and traceable to, the design and development models of the system themselves.
- Instantiations can be produced quickly and easily to reflect the current state of development.
- Change management of the argument becomes automatic.
- Consistent, reusable instantiation rules can be established, ensuring consistent and repeatable pattern instantiation.

Any approach for automating assurance argument generation requires as a minimum:

- model(s) of the required assurance argument structure - for this we use the assurance argument patterns;
- model(s) of the system - containing the information necessary to instantiate the patterns, often including models of the environment and development processes
- transformation rules to generate the output model (the assurance argument).

If we assume that we have available the required assurance argument patterns, the challenge becomes one of identifying the necessary system models, and defining a set of transformation rules. These are the focus of this paper. Section 2 discusses the system models that are required to generate an assurance argument. Section 3 discusses an approach to defining transformation rules – model weaving. Section 4 describes how model weaving can be applied to assurance cases. In section 5 we discuss related work and describe our conclusions.

2 System Models for Assurance

Assurance argument patterns can be captured using the graphical notation GSN [1]. Instantiation of assurance argument patterns involves both instantiating ‘roles’ in the argument patterns, and making instantiation choices. Roles are instantiable entities within elements of the argument pattern. They represent an abstract entity that needs to be replaced with a concrete instance appropriate for the target system. For example in Figure 1, the role within this assurance claim, represented in curled braces is ‘Function’. This entity must be replaced with the name of the relevant function of the system. In addition, argument patterns will often include multiplicity relations, where the number of required argument elements must also be determined (e.g. an entity created for each of the functions present in the system design).

Assurance argument patterns will also often represent choices for different argument approaches that may be adopted. At instantiation, the assurance claims most appropriate for the target system must be chosen from the options provided in the pattern. A more detailed example of an assurance argument pattern is provided in Figure 2.

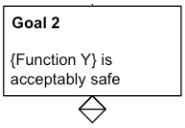


Figure 1. A GSN Argument Element Requiring Instantiation

All of these instantiation decisions are made using information about the system. The nature of the claims made in an assurance case can vary enormously between systems and domains, but in all cases there will be a requirement to include two types of argument, technical risk arguments and confidence arguments [4]. The technical arguments reason about risk reduction and the mitigation of system hazards. These will include consideration of specific design features and properties of the system. The technical argument requires consideration of design, analysis and verification artefacts. Arguments of confidence document the reasons for having confidence in the technical argument. The confidence argument will in general require consideration of the processes used to generate the development artefacts.

In most cases it is not possible to acquire all the information that is required for a complete and compelling assurance case including both technical and confidence arguments from a single model of the system. In work such as [5] it is described how it is possible to extract a lot of information required to create an assurance argument from system specifications such as AADL models. However such specifications would not contain all the information required for the assurance case. For example, although development artefacts themselves, such as safety analyses, are often integrated into such system specifications (e.g. as AADL error models), information to support a confidence argument (about the way in which those artefacts were generated) is not included (and it wouldn't be appropriate to do so). Information regarding verification is also not commonly included in such specifications. Clearly multiple models will be required to generate a complete assurance argument.

As an example we present in Figure 2 an example argument pattern that we created to form part of the assurance argument for a Distributed MILS (D-MILS) system [6]. There can be seen in this argument pattern to be a number

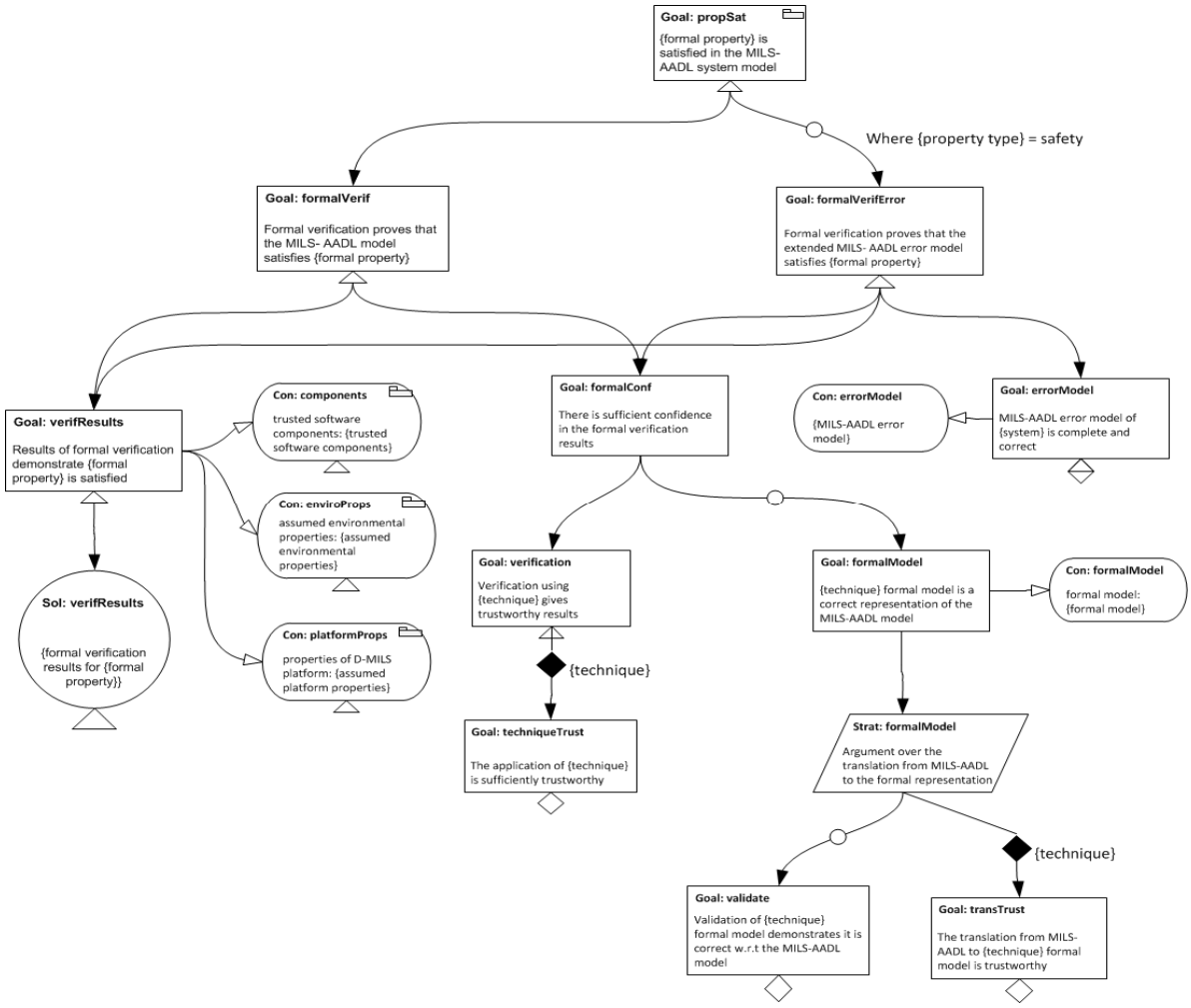


Figure 2. An example pattern for part of a D-MILS system assurance case

of roles that it was possible for us to instantiate using information extracted from an extended MILS-AADL model of the system, such as:

- formal properties (the properties to be demonstrated);
- trusted software components
- assumed platform properties.

However there can also be seen to be other claims within the argument where information will be required that is not available from the MILS-AADL model. For example the claim that the application of a particular technique to verify a formal property is sufficiently trustworthy will require information about the process for applying the technique, and about the tools used (similarly for claims regarding the translation from informal to formal representations). We obtained this information from models produced of the verification process and tool chain. Another example is the formal verification results, which are not part of the MILS-AADL model, but contained within a separate verification model.

3 Model Transformation

In the previous section we described how the instantiation of assurance argument patterns will normally require information from multiple source models. There will be (often complex) relationships between these models. Relationships will exist both between the source information models and the instantiable elements of the argument pattern models, and also between elements of the different source models. Successful pattern instantiation requires that the relationships between model elements are correctly specified.

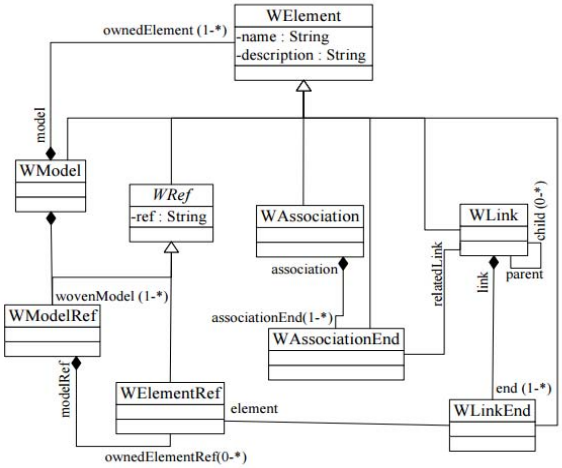


Figure 3. The weaving metamodel

Model weaving, is described in [7] as "*a method of establishing correspondences with semantic meaning between model elements*". The central concept is a weaving model which is "*a special kind of model used to save these correspondences*". Like all other models the weaving model must conform to a weaving metamodel. The basic form of the weaving metamodel, taken from [8] is shown in Figure 3. Weaving models can be created to define links between model elements. The semantics of the link can be defined for specific links in the weaving model. The weaving metamodel also includes associations that can define relationships between the links in the weaving model. In Section 4 we describe how associations and links may be used in a weaving model for an assurance case.

The weaving model that is created can then be used as the specification for model transformations to generate the output model or models from the set of source models. Model weaving can bring a number of advantages when compared with other approaches to model transformation. The weaving model specification is independent of implementation, which means that the same weaving model can be used to create multiple transformations. The semantics of the transformations in the weaving model are defined by the user. This allows much greater flexibility when applying the weaving model. In addition, as the weaving model is itself a model, it allows a seamless model-driven approach to be adopted for all aspects of the assurance case process.

4 Applying Model Weaving to Assurance Cases

In [9] we have described an approach that uses a weaving model to create an assurance argument from assurance argument pattern(s) and a set of system models. Figure 4 provides an overview of our current prototype tool that implements this approach. Below we briefly describe each of the elements.

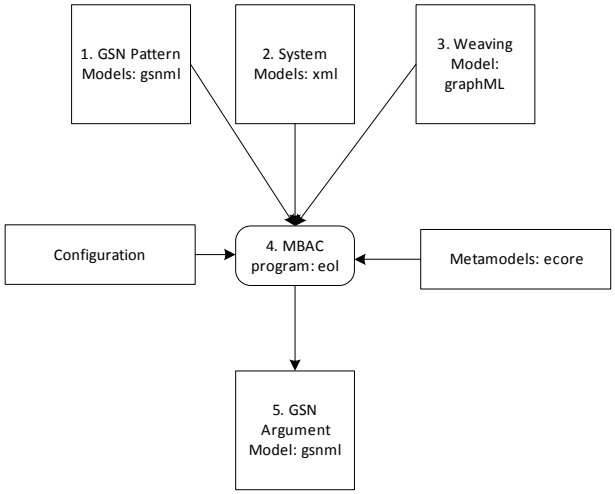


Figure 4. An implementation of a model weaving approach for assurance cases

1. The argument patterns must be provided in machine-readable format. For this we have developed a graphical editor that creates a model in an XML form from a graphical representation of the argument pattern in GSN. We refer to these files (that are compliant with the GSN metamodel) as GSNML files.

2. Any system models that conform to a defined metamodel may be taken as input.

3. The current version of the tool uses an interim solution for creating weaving models that involves creating the weaving models graphically and importing them to the tool

as graphML files. Future development of the tool will include the creation of weaving models directly from the metamodels, rather than graphically. The weaving model is represented using typed nodes and edges with properties declared to specify additional attributes such as the metamodel element type or the name of the target model. Figure 5 shows an example weaving model created in this manner. The nodes on the left hand side represent roles within the argument patterns whereas the nodes on the right hand side are elements of the source metamodels. The edges represent weaving links and associations.

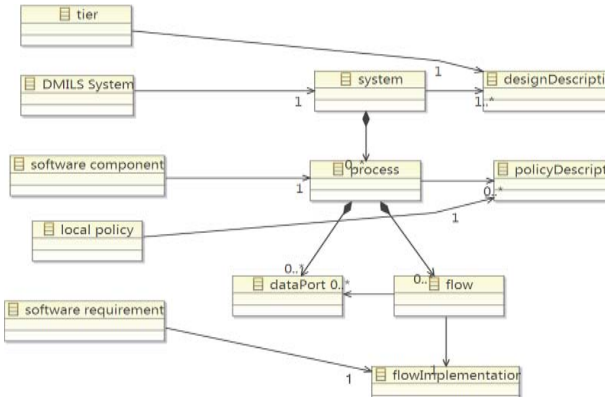


Figure 5. An example weaving model for an assurance argument pattern instantiation

Note that changes to the system models should not normally require changes to the weaving model, so long as no changes are made to the existing argument patterns and only system models conforming to the same metamodels are used. This means that changes to the system design can be quickly reflected in the assurance case.

4. The MBAC (Model Based Assurance Case) program is an Epsilon Object Language (eol) program [10] that runs on the Eclipse platform. It takes the GSNML argument pattern files, the system models and corresponding metamodels, and the weaving model as inputs. The output is a GSN argument model for the target system that has been instantiated using information extracted from the system models.

5. The argument model is generated as a GSNML file. This GSNML file can then be used to present information to the user in a number of ways. Firstly, the argument model can be represented graphically as a GSN structure. Secondly, the model can be queried in order to provide a particular view on the assurance case. For example it is possible to just select those argument elements that remain undeveloped, requiring additional support from the system developer. Finally an instantiation table can also be generated that summarises how the pattern has been instantiated in tabular form, rather than having to consult the entire argument structure.

The GSN argument model can also be used as the basis for performing verification of the assurance argument structure, as well as validation of the argument with respect to the system models. These verification and validation activities are the subject of on-going research.

5 Conclusions

It is a shared goal of many researchers [11, 12, 13] to increase automation in the generation and maintenance of assurance arguments. Our approach complements these approaches, but crucially, it does not depend on having to extract and pre-process assembly and instantiation data. By automatically extracting information directly from the design and safety analysis models themselves, a model weaving approach ensures traceability between the sources of information, e.g. in design, process and analysis models, and the assurance case. Automation in this way also has the potential to support the coevolution of system design and assurance cases.

The correct definition of the weaving model is of course crucial to the success of this approach. Although our initial work has demonstrated the feasibility of the approach, further work is required to more fully understand and model the relationships and constraints that exist between system design models (such as AADL) and other models required for the assurance case (such as process models).

Acknowledgements

This work was part funded by the European Union FP7 D-MILS project (www.d-mils.org).

References

[1] GSN Community Standard Working Group (2011), *GSN Community Standard*, Available at www.goalstructuringnotation.info/.

[2] T. Kelly and J. McDermid (1997), *Safety Case Construction and Reuse Using Patterns*, in proc. Safecomp 97, pp 55-69, Springer.

[3] R. Hawkins et. al. (2011), *Using a Software Safety Argument Pattern Catalogue: Two Case Studies*. In proc. of Safecomp 11, Springer.

[4] R. Hawkins et. al. (2011), *A New Approach to Creating Clear Safety Arguments*, In proc. of the Nineteenth Safety-Critical Systems Symposium, pp 3-23, Springer.

[5] A. Gacek et. al. (2014), *Resolute: An Assurance Case Language for Architecture Models*, In proc. of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology. pp 19-28

[6] J. Rushby (2008), *Separation and Integration in MILS* (The MILS Constitution,. Technical Report, SRI International

[7] M. Didonet et. al. (2005), *Applying generic model management to data mapping*, in proc. Bases de Données Avancées (BDA05).

[8] M. Didonet et. al. (2005), *AMW: A generic model weaver*, in proc. 1ères Journées sur l'Ingénierie Dirigée par les Modèles.

[9] R. Hawkins et. al. (2015), *Weaving an Assurance Case from Design: A Model-Based Approach*, In proc.

of 16th IEEE International Symposium on High Assurance Systems Engineering.

[10] D. Kolovos et. al. (2013), *The Epsilon Book*, available at http://www.eclipse.org/epsilon/doc/book/.

[11] E. Denney et. al. (2012), *Advocate: An Assurance Case Automation Toolset*, in proc. Workshop on Next Generation of System Assurance Approaches for Safety Critical Systems, pp 8-21.

[12] Y. Matsuno and S. Yamamoto (2013), *An implementation of GSN community standard*, In proc. of Assurance Cases for Software-Intensive Systems (ASSURE).

[13] J. Rushby (2013), *Mechanized support for assurance case argumentation*, in proc. 1st International Workshop on Argument for Agreement and Assurance (AAA 2013), Springer LNCS.

Architecture-led Requirements and Safety Analysis of an Aircraft Survivability Situational Awareness System

Peter H. Feiler

Software Engineering Institute, Fifth Ave, Pittsburgh, PA 15213; Tel: +1 412 268 7790; email: phf@sei.cmu.edu

Abstract

Software cost in mission and safety-critical systems has been escalating exponentially due to high requirement error leakage into system integration. Furthermore, system tests are designed against a large percentage of ambiguous, missing, and incomplete requirements. The Architecture Centric Virtual Integration Process (ACVIP) is being investigated by the US Army to address these challenges. ACVIP is an adaptation of the System Architecture Virtual System Integration (SAVI) approach based on the SAE Architecture Analysis & Design Language (AADL). This approach detects and removes defects through virtual integration of system models and their analysis. In this paper we describe an approach to specification of verifiable requirements and to system safety analysis that utilizes architecture models. A primary objective of this approach is to improve the quality of requirements through increased requirement coverage as well as coverage and mitigation of safety hazards.

Keywords: Virtual System Integration, Architecture Analysis & Design Language, Safety.

1 Introduction

The Software Engineering Institute® (SEI) performed an architecture-led requirement specification and safety analysis in a shadow project of the United States Army Aviation Development Directorate (ADD) on the Joint Multi Role (JMR) Technology Demonstrator effort's Joint Common Architecture Demonstration (JCA Demo) Project [1] to investigate and mature the Architecture Centric Virtual Integration Process (ACVIP). ACVIP is a DoD process fashioned after System Architecture Virtual Integration (SAVI) [2] performed by a consortium of aerospace organizations. Like SAVI, the purpose of the ACVIP is to address the affordability and associated risks of developing complex software intensive systems through early virtual integration and analysis before implementation.

Architecture-led requirement specification (ALRS) addresses the problem of a high percentage of ambiguous, missing, and incomplete requirements found in textual requirement documents that result in costly rework later in development. It improves the quality of requirements by assuring better coverage of requirements along two dimensions: coverage of interactions and of quality attributes. Architecture-led safety analysis (ALSA) assures improved coverage of safety hazards through a fault propagation ontology and allows for automation of currently labor-intensive best safety analysis practices, e.g., SAE ARP4761.

2 Architecture Led Requirements Specification

The ALRS process utilizes the AADL and encompasses the eleven step process outlined in the Federal Aviation Administration (FAA) Requirements Engineering Management Handbook [3]. ALRS adapts the CPRET [4] representation of a system defined by the *Association Française d'Ingénierie Système* which is shown graphically in Figure 1.

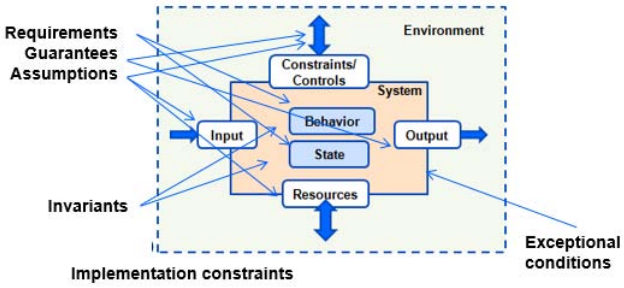


Figure 1 Elements of a System Specification

In the ALRS process a user models a system in its operational context as an AADL model of interacting systems. An explicit model of these interacting systems guides the user to specifying requirements regarding each of these system interactions in terms of input assumptions, output guarantees, invariants on system state and behavior, as well as assumptions about resources being utilized, and interactions with supervisory capabilities.

When used in the context of an existing requirement document, users map the requirements to an AADL model. This mapping helps the user to quickly identify any gaps in the set of requirements. It also lets the user see whether a requirement section cover one or more system components.

ALRS utilizes utility trees from a Quality Attribute Workshop (QAW) [5] or an Architecture Tradeoff Analysis

Method (ATAM) [6] to provide a framework for achieving coverage of non-functional properties, also known as operational quality attributes. Prioritization of the utility tree leafs driven by mission goals help the user ensure that critical requirements are well-specified. Such a utility tree is shown in Figure 2.

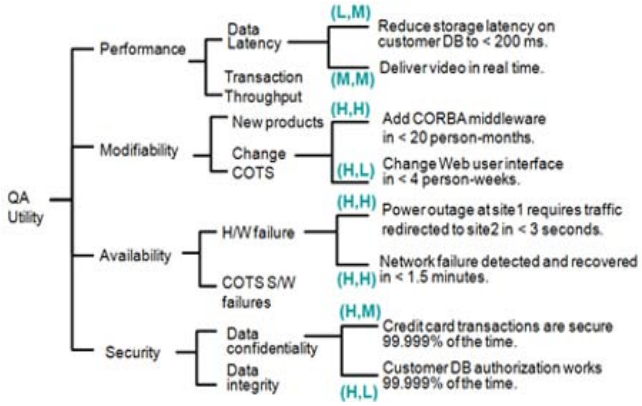


Figure 2 Quality Attribute Utility Tree

Early in the development process the SEI team captured requirement information from the JCA Demo BAA and Stakeholder and Systems Requirements documents of the aircraft survivability situational awareness (ASSA) system as well as UML models made available to suppliers of a data correlation and fusion system. This analysis identified shortcomings in the system-level and component-level requirements. They included inconsistencies, and missing requirement information in the original documents, as well as defects related to safety, latency, and timing / resource utilization. This was achieved by modeling the system in its operational context as well as the functional and the system architecture of the ASSA itself. The resultant architecture model was generalized into an aircraft survivability situational awareness (ASSA) system, creating a reusable reference architecture for the domain of use.

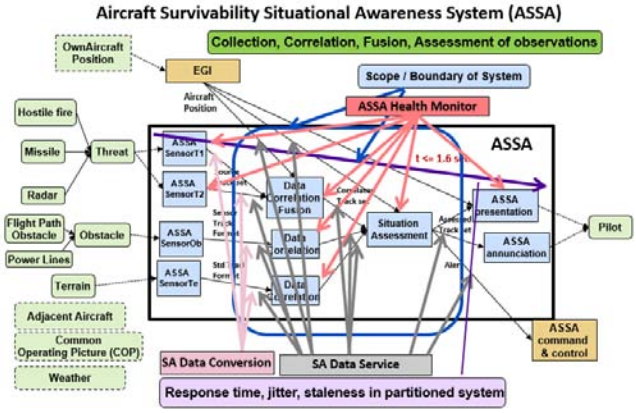


Figure 3 Layered Architecture of ASSA System

This ASSA system incorporates several functional services. Figure 3 shows the functional architecture of ASSA with a clear delineations of its interface with the operational environment. In addition it shows three infrastructure services. Two services are provided in a layer below the situational awareness system, i.e., the data conversion service, and the data management service. The third service, a health monitor, resides in a layer above the situational awareness system to detect and report any exceptional conditions in the operation.

The resultant functional architecture also became the basis for quantitative analysis of the ASSA early in development, e.g., pre-PDR. As Figure 3 shows, the model included end-to-end flow specifications of a critical flow to represent response time requirements. It also captures a UML sequence diagram from the original documentation as an analyzable interaction protocol across ARINC653 partitions. The latency analysis capability of OSATE2 informed us of the latency overhead contributed by this protocol, and its effect on the critical flow, i.e., that in the best circumstances the requirement can barely be met.

3 Architecture Led Safety Analysis

The ALSA process builds on the AADL created for the ASSA during the requirement specification process. The user annotates an AADL model with fault information utilizing an error propagation ontology as illustrated graphically in Figure 4. The error propagation ontology addresses issues of service omission, commission, value, timing, rate, sequence, replication, concurrency, authorization, and authentication errors. Users can adapt this ontology to commonly used hazard guide words, such as loss of power. The propagation paths between system components are derived from the architecture specification itself.

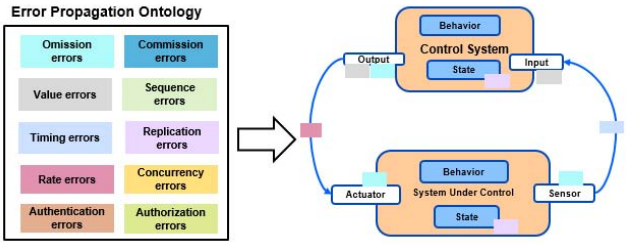


Figure 4 Identification of Hazard Sources and Impact

This process leverages the AADL Error Model Version 2 (EMV2) Annex [7] to support SAE ARP-4761 [8] best system safety analysis practices, such as an FHA, FMEA and FTA. The analysis models, such as a fault tree, are generated from the annotated AADL model, and then processed by a FTA tool. In the case of FHA and FMEA the respective reports are generated directly from the annotated AADL model – as shown in Figure 5. In the SAVI initiative the SEI recently demonstrated how the SAE ARP-4761 process can be supported by an AADL model annotated with fault information using the Error Model Annex standard for AADL on an aircraft wheel braking system. FHA, FMEA, and FTA reports as well reliability/availability analysis reports have been generated from safety analysis performed with such a model.

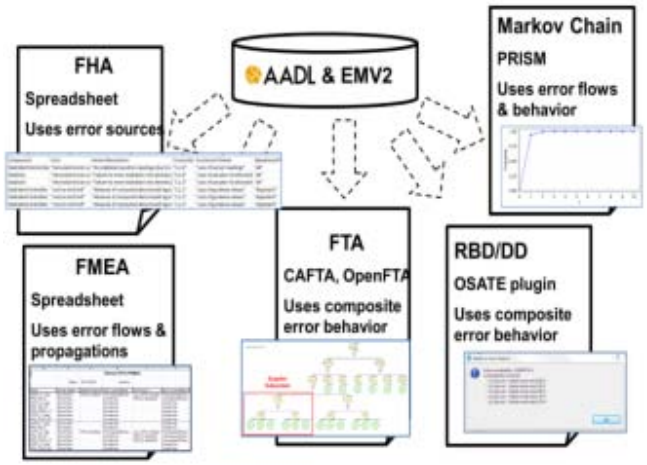


Figure 5 Safety Analyses from Annotated AADL Models

In the original safety analysis practice ASSA was assigned a design assurance level E with respect to flight worthiness. However, since aircraft does get lost due to enemy threats, obstacles, and terrain variation, we considered it a critical component that requires the attention of a safety analysis. ALSA allowed us to identify the safety hazards of ASSA in its operational context and systematically identify hazard contributors. In addition to complete failure of providing the ASSA service, the hazards considered included providing false information such as false positives in the form of alerting the pilot of threats and obstacles that do not exist, false negatives such as not alerting the pilot when these threats and obstacles exist. In addition the timeliness of information was taken into account, i.e., how much information delay is acceptable to the pilot. Subsequent to citing the hazards, the potential error sources were systematically identified that can propagate as one of the identified hazard categories to the pilot. A fault ontology provided as part of the AADL Standard Error Model annex was used as a checklist of fault propagation categories to consider in the process.

The insights from this analysis lead to a set of derived safety requirements for the health monitoring system that were lacking in the original System Requirement document. The original requirement document discussed detection of non-operational sensors and transitioning to operational mode as long as one sensor is operational. ALSA leads to a clear identification of all system components being monitored and the appropriate health status to be reported to the pilot, all derived requirement on the health monitoring system.

4 JCA Demo ACVIP Analysis Findings and Lessons Learned

Previous studies have shown that peer review is a very cost-effective means of defect detection, partly because it was the only traditional method that could be applied in early development phases. The ACVIP researcher's experience is that many defects were detected during model development even before analysis tools were applied. This is achieved by mapping terms in the document into concepts expressed by AADL. Users quickly realize different terms used in different sections of the documents for the same concepts, and conflicting statements about specific attributes of model elements, e.g., two different numbers for range of operation. Strong typing in AADL ensures that interactions between virtually integrated system components are consistent, e.g., that measurement units and interchange protocols are used consistently. In other words, the rigor of the AADL focuses attention on ambiguous and incomplete elements of a natural language document and eliminates potential system integration problems early in the process. This is consistent with earlier reports that a significant benefit of modeling is more precise specification; many defects are found during the model development phase [9].

Earlier studies showed that providing reviewers with structured guidelines (often called reading guidelines or techniques in the inspection literature) improved the quality of reviews. In model-based engineering, the model development task could be viewed as a particularly well-structured review method [10].

The ACVIP related goals for JMR Mission Systems Architecture Demonstrations (MSAD) such as the JCA Demo are to identify, validate, mature and transition methods and tools to support an architecture centric virtual integration process. This exercise also generated new modeling guidelines and tool requirements (as well as bug reports for tool developers and errata for the AADL standards committee).

The ACVIP researchers provided reports citing around 85 findings, 70 that were attributed to requirements analyses and 15 to timing analyses that have been rolled up in the JCA Demonstration Final Report and summarized in [11]. Some notable areas identified by the ACVIP team included:

- Lack of a specification of staleness for the data.
- No identification of end-to-end timing requirement for specific types of threats and obstacles.
- Partition schedule not meeting ARINC 653 scheduling rules.
- Non-clarity in pull protocol between data correlation/fusion and SA Data Service.
- Impact of cross partition pull protocols on end-to-end latency affects end-to-end timing requirements.
- Data storage requirement for the SA Data Service not specified.
- Ambiguity on the Operational State under timeout conditions.
- Lack of a requirement for the number of source tracks the aircraft survivability sensor provides.
- Potential of data integrity issues in time-sensitive track data that manifests itself as noisy data.
- Multiple sensor stream rates may have implications on integration.

- Inconsistency in the observation radius and alert threshold for threats and obstacles.
- Potential memory leaks in SA Data Service.
- Ambiguity in the requirement to correlate 50 source tracks within 1 second and concern over meeting the requirement.

Some of these issues were also cited by the contractors independent of the ACVIP researchers. The development team was able to confirm several of these and other findings by ACVIP during development and in integration testing. The findings by the ACVIP team demonstrated that in a real program that these issues would have been identified and corrected even prior to solicitation which could have led to a cost savings and / or development schedule reduction.

Conclusion

ACVIP is an architectural centric model based approach that will revolutionize the way in which we analyze our systems. Results of the JCA Demo ACVIP Shadow effort demonstrated that ACVIP has potential to provide strong architectural analysis to identify and aid in the early resolution of issues. AADL is being used in many company and organization research efforts. ACVIP and its guidance, tools, and processes are going through maturation and require further refinements and maturation to be effective for future DoD acquisition of aviation mission computing systems. JMR Mission Systems Architecture Demonstrations will continue to work with the ACVIP researchers and ensure that the exercise, documentation and lessons learned mature these processes and tools so that they can effectively be used by avionics and systems engineers in the future. Industry and Government need to work together to improve ACVIP so that future development / integration efforts can benefit from early virtual integration, validation and verification.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. This material has been approved for public release and unlimited distribution. Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University. DM-0002390

References

[1] Department of the Army, Army Contracting Command (2014), *A Joint Multi-Role Technology Demonstrator (JMR TD) Joint Common Architecture Demonstration (JCA Demo) Broad Agency Announcement (BAA),* Location ACC-RSA-AATD-(SPS). Solicitation Number W911W614R000002.

[2] Aerospace Vehicle Systems Institute. http://savie.avsi.aero. [Online]

[3] DOT/FAA/AR-08/32 (2009), *Requirements Engineering Managmeent Handbook.*

[4] Association Française d'Ingénierie Système. CPRET: System Process as Constraints, Products, Resources, input Elements and Transformations. [Online] http://en.wikipedia.org/wiki/Process_%28engineering%29#CPRET.

[5] CMU-SEI. Quality Attribute Workshop. [Online] http://www.sei.cmu.edu/architecture/tools/establish/qaw.cfm.

[6] CMU SEI. Architecture Tradeoff Analysis Method [Online] http://www.sei.cmu.edu/architecture/tools/establish/atam.cfm

[7] SAE International, AS-2C (2015), *Architecture Analysis and Design Language (AADL) Annex Volume 1 Annex E: Error Model Annex,* AS 5506/1A.

[8] SAE International, SAE ARP-4761 (1996), *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment.*

[9] E. M. Clark, Jeannette M. Wing (1996), *Formal Methods: State of the Art and Future Directions, ACM Computing Surveys.*

[10] O. Laitenberger (2002), *A Survey of Software Inspection Technologies, Handbook on Software Engineering and Knowledge Engineering.*

[11] A. Boydston,, P. Feiler, S. Vestal, B. Lewis (2015), *Joint Common Architecture (JCA) Demonstration Architecture Centric Virtual Integration Process (ACVIP) Shadow Effort,* AHS 71st Annual Forum, Virginia Beach, Virginia, May 5–7.

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 in Sweden

attn. Rei Stråhle
Rimbogatan 18
SE-753 24 Uppsala
Sweden
Phone: +46 73 253 7998
Email: rei@ada-sweden.org
URL: www.ada-sweden.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