MAPS

NUMMER 51 • VOORJAAR 2021

NEDERLANDSTALIGE TEX GEBRUIKERSGROEP

Voorzitter
Hans Hagen
voorzitter@ntg.nl

Secretaris
Taco Hoekwater
secretaris@ntg.nl

Penningmeester
Ferdy Hanssen
penningmeester@ntg.nl

Bestuursleden
Frans Goddijn
Pieter van Oostrum

Postadres
Nederlandstalige TEX Gebruikersgroep
Baarsjesweg 268-I
1058 AD Amsterdam

ING bankrekening
IBAN: NL53INGB0001306238
BIC: INGBNL2A

E-mail bestuur
ntg@ntg.nl

E-mail MAPS redactie
maps@ntg.nl

WWW
www.ntg.nl

De **Nederlandstalige TEX Gebruikersgroep (NTG)** is een vereniging die tot doel heeft de kennis en het gebruik van TEX te bevorderen. De NTG fungeert als een forum voor nieuwe ontwikkelingen met betrekking tot computergebaseerde documentopmaak in het algemeen en de ontwikkeling van 'TEX and friends' in het bijzonder. De doelstellingen probeert de NTG te realiseren door onder meer het uitwisselen van informatie, het organiseren van conferenties en symposia met betrekking tot TEX en daarmee verwante programmatuur.
De NTG biedt haar leden ondermeer:

- Tweemaal per jaar een NTG-bijeenkomst.
- Het NTG-tijdschrift MAPS.
- De 'TEX Live'-distributie op DVD/CDROM inclusief de complete CTAN software-archieven.
- Verschillende discussielijsten (mailing lists) over TEX-gerelateerde onderwerpen, zowel voor beginners als gevorderden, algemeen en specialistisch.
- De FTP server `ftp.ntg.nl` waarop vele honderden megabytes aan algemeen te gebruiken 'TEX-producten' staan.
- De WWW server `www.ntg.nl` waarop algemene informatie staat over de NTG, bijeenkomsten, publicaties en links naar andere TEX sites.
- Korting op (buitenlandse) TEX-conferenties en -cursussen en op het lidmaatschap van andere TEX-gebruikersgroepen.

Lid worden kan door overmaking van de verschuldigde contributie naar de NTG-giro (zie links); vermeld IBAN zowel als SWIFT/BIC en selecteer shared cost. Daarnaast dient via `www.ntg.nl` een informatieformulier te worden ingevuld. Zonodig kan ook een papieren formulier bij het secretariaat worden opgevraagd.
De contributie bedraagt € 35. Voor studenten geldt een tarief van € 18. Dit geeft alle lidmaatschapsvoordelen maar *geen stemrecht.* Een bewijs van inschrijving is vereist. Een gecombineerd NTG/TUG-lidmaatschap levert een korting van 10% op beide contributies op. De prijs in euro's wordt bepaald door de dollarkoers aan het begin van het jaar. De ongekorte TUG-contributie is momenteel $105.

Afmelding kan met ingang van het volgende kalenderjaar door opzegging per e-mail aan de penningmeester.

MAPS bijdragen kunt u opsturen naar `maps@ntg.nl`, bij voorkeur in LATEX- of ConTEXt formaat. Bijdragen op alle niveaus van expertise zijn welkom.

Productie. De Maps wordt gezet met behulp van een LATEX class file en een ConTEXt module. Het pdf bestand voor de drukker wordt aangemaakt met behulp van pdftex 1.40.20 en luatex 1.13.0 draaiend onder MacOS X 11.2. De gebruikte fonts zijn Linux Libertine, het niet-proportionele font Inconsolata, schreefloze fonts uit de Latin Modern collectie, en de Euler wiskunde fonts, alle vrij beschikbaar.

TEX is een door professor Donald E. Knuth ontwikkelde 'opmaaktaal' voor het letterzetten van documenten, een documentopmaaksysteem. Met TEX is het mogelijk om kwalitatief hoogstaand drukwerk te vervaardigen. Het is eveneens zeer geschikt voor formules in mathematische teksten.
Er is een aantal op TEX gebaseerde producten, waarmee ook de logische structuur van een document beschreven kan worden, met behoud van de letterzet-mogelijkheden van TEX. Voorbeelden zijn LATEX van Leslie Lamport, $\mathcal{AMS}$-TEX van Michael Spivak, en ConTEXt van Hans Hagen.

Contents

Redactioneel

Door middel van de Maps willen we u op de hoogte houden van ontwikkelingen, ook om daarmee onze leden te danken voor hun trouwe steun aan de TEX ontwikkelaars. Verder bieden we ruimte aan lezers die anderen laten delen in hun ervaringen met TEX, MetaPost, fonts en aanverwanten. Aarzel dus niet ons artikelen te sturen. Een halve pagina is al heel leuk, meer mag ook, graag zelfs. Het hoeft geen ‚zware kost' te zijn want het is voor lezers bijvoorbeeld al heel interessant te lezen hoe anderen TEX gebruiken. Dus een artikeltje als „dit doe ik met TEX, zo doe ik dat en nu kun jij het ook" is zeer welkom!

Hoewel het internet tegenwoordig een belangrijke bron van informatie is, blijft papier een functie vervullen binnen de vereniging. Dat past immers bij TEX!

Veel leesplezier,

Uw redactie

Waarom is een NTG lidmaatschap belangrijk?

TEX bestaat bijna 40 jaar en het ziet er niet naar uit dat het systeem snel zal verdwijnen. Oorspronkelijk was het alleen beschikbaar op universiteiten, later kon men het ook thuis installeren. Verschillende ‚journals' worden vormgegeven in TEX en op universiteiten is het nog steeds populair.

Het begon met TEX, geschreven door Donald Knuth. Later kwamen er wat extensies onder de vlag ε-TEX. De succesvolle variant met ingebouwd backend PDFTEX werd na verloop van tijd de standaard. Toen UNICODE opgang maakte kwam XETEX erbij, en wat later begon de ontwikkeling van LUATEX. Parallel aan deze ontwikkeling zijn fonts ontwikkeld, is MetaPost uitgekristalliseerd en zijn macro-pakketten als LATEX en ConTEXt ontstaan die een groot aantal talen en scripts ondersteunen.

De hele TEX beweging kan worden getypeerd met woorden als *originaliteit*, *solidariteit*, *continuïteit*, *ondersteuning* en *ontwikkeling*. Dat alles vinden we als gebruikers heel normaal. Vanzelfsprekend is het feit dat we al zo lang aanwezig zijn mede het gevolg van de sympathieke Donald Knuth. Rond hem ontstond de eerste gebruikersgroep TUG, maar al snel verenigden gebruikers zich in groepen, in de regel per taalgebied, met als doel ondersteuning van het taalgebied en de kenmerkende (lokale) typografie. De gebruikersgroepen organiseren bijeenkomsten, publiceren tijdschriften en hosten websites en mailing lijsten.

De verschillende programma's (ook wel ‚engines' genoemd) worden al sinds jaren verspreid door de gebruikersgroepen, de laatste decennia op CD's and DVD's. De Nederlandstalige Gebruikersgroep heeft daarbij een pioniersrol gespeeld. De programma's worden vergezeld van een uitgebreide collectie fonts en macros.

Tegenwoordig vindt de communicatie vooral plaats via internet, maar in het verleden (en met enige regelmaat nog steeds) kwamen ontwikkelaars samen op (internationale) bijeenkomsten. Regelmatig zijn door gebruikersgroepen projecten geïnitieerd en ondersteund. Op dit moment draagt de NTG substantieel bij aan de ontwikkeling van moderne fonts.

Wat gebeurt er als de gebruikersgroepen wegvallen? Dat is niet met zekerheid te zeggen, maar het is een feit dat er heel veel gebruikers zijn die hun hele leven lang het TEX ecosysteem gebruiken. Zij kunnen dankzij de gebruikersgroepen vooralsnog verzekerd zijn van continuïteit en stabiliteit. En dat zonder de tegenwoordig alom tegenwoordige reclame (zoals in apps), zonder lock-in op een bepaald hardware platform, onafhankelijk van een besturingssysteem, en natuurlijk zonder dringende uitnodiging om een betaalde pro-versie te gebruiken of deel te nemen aan een of ander ‚plan'.

Zal de rol van de vereniging veranderen? Natuurlijk! Publicaties worden bijvoorbeeld wat zeldzamer en concentreren zich op ontwikkelingen. Bijeenkomsten worden kleinschaliger en trekken vooral actieve TEXers wat weer bijdraagt aan de kwaliteit, volledigheid en continuïteit van distributies. Mailing lijsten en websites worden tegenwoordig aangevuld met forums en blogs. Er zijn nog steeds internationale bijeenkomsten.

De financiële positie van de NTG is solide, ondanks het feit dan het ledental wat afneemt en vergrijst. Vooralsnog wordt er gebruik gemaakt van (door leden) gesponsorde sites voor het hosten van de distributies en archieven. Als de font projecten zijn afgerond, is er weinig reden om geld te reserveren voor ontwikkeling. Om die reden zijn de kosten niet gestegen en kan het lidmaatschap laagdrempelig worden gehouden.

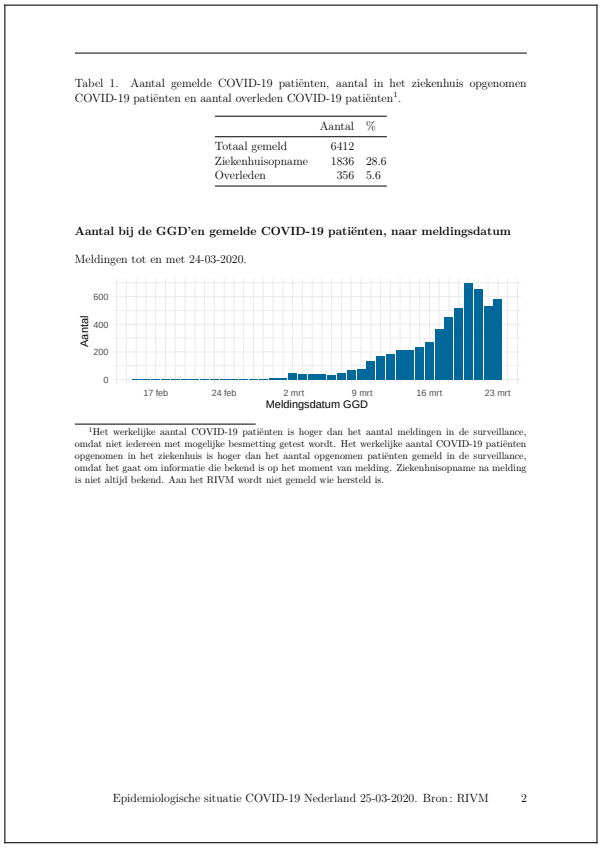
Lid zijn van de vereniging is afgezien van een garantie voor de beschikbaarheid van distributies ook een signaal aan gebruikers dat die garantie er vooralsnog is. Niet alles kan via internet, soms moeten we elkaar gewoon ontmoeten. En het helpt als ontwikkelaars weten dat er gebruikers zijn die zich willen verbinden aan het lot van TEX. Kortom, als de verenigingen zouden verdwijnen dan ontstaat denk ik snel een versnipperde en instabiele situatie. En dat willen we niet, toch?

Hans Hagen, voorzitter NTG

Proud or ashamed

In TUGboat, Volume 41 (2020), No. 1, the user group president Boris Veytsman ends his introduction with “When I read papers on COVID, I habitually check how they are typeset. When I see TEX I feel a pride that somehow our efforts contributed to the common task.” I have to admit that I seldom go on the web searching for such content, but I did run into a publication from the Dutch RIVM, the institute that deals with matters like COVID.

Now, it must be said, that when I was a third the age that I’m now, I actually sometimes bought the nicely typeset and printed copies of government publications. They had at that time their own typesetting and printing facilities and many such documents (like those about long term environmental planning) always looked very nice. It did make me kind of feel proud. But those times are gone, and nowadays we often get pretty mediocre stuff. There are however exceptions, like the MH17 report named : ‘Report MH17 crash’, which looks quite okay. It was not made by TEX but with InDesign, but could as well have been done with TEX.



Tabel 1. Aantal gemelde COVID-19 patiënten, aantal in het ziekenhuis opgenomen COVID-19 patiënten en aantal overleden COVID-19 patiënten[1].

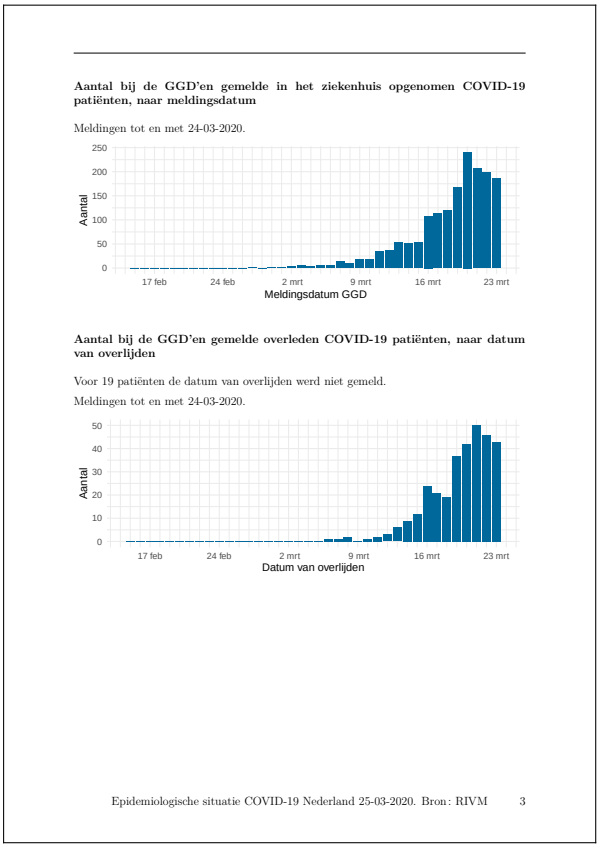
	Aantal	%
Totaal gemeld	6412	
Ziekenhuisopname	1836	28.6
Overleden	356	5.6

Aantal bij de GGD’en gemelde COVID-19 patiënten, naar meldingsdatum

Meldingen tot en met 24-03-2020.

[1]Het werkelijke aantal COVID-19 patiënten is hoger dan het aantal meldingen in de surveillance, omdat niet iedereen met mogelijke besmetting getest wordt. Het werkelijke aantal COVID-19 patiënten opgenomen in het ziekenhuis is hoger dan het aantal opgenomen patiënten gemeld in de surveillance, omdat het gaat om informatie die bekend is op het moment van melding. Ziekenhuisopname na melding is niet altijd bekend. Aan het RIVM wordt niet gemeld wie hersteld is.

Page 2



Aantal bij de GGD’en gemelde in het ziekenhuis opgenomen COVID-19 patiënten, naar meldingsdatum

Meldingen tot en met 24-03-2020.

Aantal bij de GGD’en gemelde overleden COVID-19 patiënten, naar datum van overlijden

Voor 19 patiënten de datum van overlijden werd niet gemeld.

Meldingen tot en met 24-03-2020.

Page 3

Figure 1.

Tabel 2. Leeftijdsverdeling van bij de GGD'en gemelde COVID-19 patiënten, van in het ziekenhuis opgenomen COVID-19 patiënten en van overleden COVID-19 patiënten[2].

Leeftijdsgroep	Totaal	%	Ziekenhuisopname	%	Overleden	%
Totaal gemeld	6412		1836		356	
0-4	23	(0.4)	10	(0.5)	0	(0.0)
5-9	6	(0.1)	1	(0.1)	0	(0.0)
10-14	30	(0.5)	4	(0.2)	0	(0.0)
15-19	42	(0.7)	4	(0.2)	0	(0.0)
20-24	119	(1.9)	5	(0.3)	0	(0.0)
25-29	322	(5.0)	23	(1.3)	0	(0.0)
30-34	330	(5.1)	28	(1.5)	0	(0.0)
35-39	283	(4.4)	20	(1.1)	0	(0.0)
40-44	282	(4.4)	27	(1.5)	0	(0.0)
45-49	460	(7.2)	87	(4.7)	0	(0.0)
50-54	529	(8.3)	115	(6.3)	2	(0.6)
55-59	602	(9.4)	131	(7.1)	2	(0.6)
60-64	487	(7.6)	157	(8.6)	7	(2.0)
65-69	484	(7.5)	226	(12.3)	24	(6.7)
70-74	562	(8.8)	265	(14.4)	33	(9.3)
75-79	599	(9.3)	275	(15.0)	70	(19.7)
80-84	574	(9.0)	233	(12.7)	105	(29.5)
85-89	426	(6.6)	167	(9.1)	80	(22.5)
90-94	175	(2.7)	47	(2.6)	23	(6.5)
95+	53	(0.8)	6	(0.3)	9	(2.5)
Niet gemeld	23	(0.4)	4	(0.2)	1	(0.3)
NA	1	(0.0)	1	(0.1)	NA	NA

Tabel 3. Man-vrouwverdeling van bij de GGD'en gemelde COVID-19 patiënten, van in het ziekenhuis opgenomen COVID-19 patiënten en van overleden COVID-19 patiënten.

Geslacht	Totaal	%	Ziekenhuisopname	%	Overleden	%
Totaal gemeld	6412		1836		356	
Man	3181	(49.6)	1155	(62.9)	226	(63.5)
Vrouw	3199	(49.9)	673	(36.7)	128	(36.0)
Niet gemeld	32	(0.5)	8	(0.4)	2	(0.6)

[2]Het werkelijke aantal COVID-19 patiënten is hoger dan het aantal meldingen in de surveillance, omdat niet iedereen met mogelijke besmetting getest wordt. Het werkelijke aantal COVID-19 patiënten opgenomen in het ziekenhuis is hoger dan het aantal opgenomen patiënten gemeld in de surveillance, omdat het gaat om informatie die bekend is op het moment van melding. Ziekenhuisopname na melding is niet altijd bekend. Aan het RIVM wordt niet gemeld wie hersteld is.

Page 4

Tabel 4a. Aantal bij de GGD'en gemelde COVID-19 patiënten, in het ziekenhuis opgenomen COVID-19 patiënten en overleden COVID-19 patiënten met onderliggende aandoeningen en/of zwangerschap[3].

Comorbiditeit	Totaal	%	Ziekenhuisopname	%	Overleden	%
Totaal gemeld	6412		1836		356	
Ja	1748	(27.3)	974	(53.1)	191	(53.7)
Geen onderliggende aandoening	1411	(22.0)	322	(17.5)	17	(4.8)
Niet gemeld	3253	(50.7)	540	(29.4)	148	(41.6)

Tabel 4b. Voorkomende onderliggende aandoeningen[4] en/of zwangerschap van bij de GGD'en gemelde COVID-19 patiënten en in het ziekenhuis opgenomen COVID-19 patiënten en overleden COVID-19 patiënten[3]. Meerdere onderliggende aandoeningen kunnen gemeld zijn per patiënt. De meest voorkomende onderliggende aandoeningen zijn Cardiovasculaire aandoening, Chronische longaandoening en Diabetes.

Comorbiditeit	Totaal	%	Ziekenhuisopname	%	Overleden	%
Totaal voorkomende aand.	2510		1455		319	
Zwangerschap	27	(1.1)	4	(0.3)	0	(0.0)
Cardio-vasc. aand. en hypertensie	690	(27.5)	437	(30.0)	84	(26.3)
Diabetes	310	(12.4)	213	(14.6)	49	(15.4)
Leveraandoening	10	(0.4)	5	(0.3)	1	(0.3)
Chron. (neuro)musculaire aand.	81	(3.2)	35	(2.4)	9	(2.8)
Immuundeficiëntie	44	(1.8)	23	(1.6)	1	(0.3)
Nieraandoening	113	(4.5)	69	(4.7)	23	(7.2)
Chron. longaandoening	399	(15.9)	234	(16.1)	47	(14.7)
Maligniteit	172	(6.9)	111	(7.6)	23	(7.2)
Overig	664	(26.5)	324	(22.3)	82	(25.7)

[3]Het werkelijke aantal COVID-19 patiënten is hoger dan het aantal meldingen in de surveillance, omdat niet iedereen met mogelijke besmetting getest wordt. Het werkelijke aantal COVID-19 patiënten opgenomen in het ziekenhuis is hoger dan het aantal opgenomen patiënten gemeld in de surveillance, omdat het gaat om informatie die bekend is op het moment van melding. Ziekenhuisopname na melding is niet altijd bekend. Aan het RIVM wordt niet gemeld wie hersteld is

[4]Totaal voorkomende aand. = Totaal voorkomende aandoeningen, Card. vasc. aand. = Cardio-vasculaire aandoening, Chron. (neuro)musculaire aandoening = Chronische neurologische of neuromusculaire aandoening, Chron. longaandoening = Chronische longaandoening

Page 5

Figure 2.

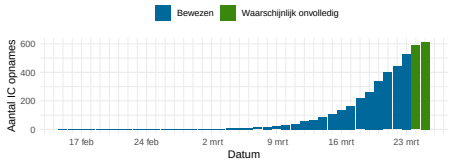
Back to COVID and the RIVM, I was actually quite horrified by a document I saw online, named 'Epidemiologische situatie COVID-19'. The document properties gave no indication how it has been produced. The producer is Acrobat Pro but that can as well be a backend job. The first page looks like your average word processor text, some colourful graphics might have been produced by a desktop publishing program, and the rest (running text and some tables) scream TEX.

Now it's not my habit to criticise documents this way but in the perspective of Boris' remark I think the TEX community should be honest about reality: there are lots of nice looking documents around made by TEX but maybe even more horrible looking ones. The fact that we have TEX doesn't mean that, to quote him again, "Following Knuth, we use rational methods to create beautiful pages in service of presentation of beautiful thoughts.". We can also say "Ignoring Knuth, we use irrational methods to create awful pages in service of presentation of whatever we want to look scientific.".

On the second page we notice a couple of things. There is hardly any text and most is actually in a footnote that ends up at the end of the text and not at the bottom of the page. Yes, because TEX can do footnotes, users love them. This is a typical example of where a footnote could have been running text. The table caption is at the top of the table, but the graphic has an unnumbered title. The spacing around the table is kind of weird as is the spacing around the graphics. The first graphic on the third page probably fits on the second page.

Totaal aantal bewezen COVID-19 patiënten per datum op de Nederlandse intensive care afdelingen

Wegens drukte op de IC is er mogelijk een vertraging van 2 a 3 dagen in de data-aanlevering, hieronder in groen aangegeven.



Bron: Nationale Intensive Care Evaluatie (NICE)
Voor meer figuren, zie www.stichting-nice.nl

Page 6

Tabel 5. Aantal bij de GGD'en gemelde COVID-19 patiënten per provincie[5].

Provincie	Aantal	%
Totaal gemeld	6412	
Drenthe	60	0.9
Flevoland	85	1.3
Friesland	53	0.8
Gelderland	695	10.8
Groningen	84	1.3
Limburg	786	12.3
Noord-Brabant	1915	29.9
Noord-Holland	870	13.6
Overijssel	326	5.1
Utrecht	574	9.0
Zeeland	77	1.2
Zuid-Holland	887	13.8

[5]Het werkelijke aantal COVID-19 patiënten opgenomen in het ziekenhuis is hoger dan het aantal opgenomen patiënten gemeld in de surveillance, omdat het gaat om informatie die bekend is op het moment van melding

Page 7

Figure 3.

Page four and five also have tables and of course footnotes. The running text is moved to the table captions. In the tables we see parenthesis around the percentage entries and the percentage symbol is not centred. Again, like footnotes, TEXies often claim that the program can make nice tables but when a macro package doesn't help (or steer) them doing just that such a claim is useless. Last year I attended a session where some hundred students graduated and the teachers showed some results from papers on the beamer. Most of the tables (the showcase I presume) looked horrible: bad spacing and lots of rules. All done in TEX, but not showing 30 years of progress in usage. Lucky me that my colleague and I were probably the only adults in the room that immediately recognised TEX, so no harm was done to its reputation.

To the tables in the COVID document we can point out that the captions could have been centred and maybe in bold and there is no apparent reason for these tables to run into the margin. There is also no reason for the bad rendering of the paragraph in the footnotes. Footnote number 4 looks like some "text done in math mode" job to me. Pages six and seven could have been one page, and in fact the whole document could have been at least one page less.

Now, one can blame the composer of the document but we can equally well blame the software. If an argument for using TEX is the structured approach that leads to quality, then here it failed. Actually, I see no properties that give reason for even using TEX. A word document might even look better.

Now the reason for even going in such detail about an otherwise irrelevant aspect of presenting data, is that I noticed that in presenting information about COVID to

a countries inhabitants, some governments actually have a very well designed web presence: nicely designed websites, adaptive graphics, tables with nice colours and spacing. There is absolutely no reason for not having a variant in print done with TEX. Actually, there is probably no program like TEX that can produce a flow of documents with a high degree of consistency in rendering.

I think that after many decades of TEX its qualitative properties are seldom exposed by publishers (or equivalents). It's common users who make the beautiful documents. It's them who sit behind the screen and spend time, maybe even struggling, experimenting, trialing and erroring in order to come to something that they think looks beautiful. When you use TEX you enter a feedback loop. You have to spend time, and if you don't want to do that, don't use it. It's those users who gets inspired by Don Knuths legacy, nice looking manuals, maybe articles in user group journals, posts on mailing lists, help on forums. Forget about the publishers, they seldom care. Forget about institutions that demand usage of TEX for e.g. thesis and reports using some decades old template. It's the freedom of users that produce nice stuff. And often these documents don't scream TEX. I bet that when one can recognise a TEX document, it often is also a not so nice looking document. Don Knuths work is of course the exception to this rule. I therefore end with quoting Boris again: "TEX was born from the striving for beauty and rationality." With that I completely agree.

You can find the mentioned texts at

- ☐ www.tug.org/TUGboat/Pres/tb127pres.pdf
- ☐ www.onderzoeksraad.nl/en/media/attachment/2018/7/10/debcd724fe7breport_mh17_crash.pdf
- ☐ www.rivm.nl/sites/default/files/2020-03/Epidemiologische situatie COVID-19 2025 20maart 202020 20Nieuw.pdf

Hans Hagen

Lost in fonts

Nowadays it is rather common to use a (wide font) OpenType aware engine but for quite a while the eight bit fonts dominated the TEX landscape. The pdfTEX engine could actually use wide TrueType fonts, but that was more a hack: only an eight bit subset could be used. This trick permitted for instance using large cjk fonts in TrueType format by splitting the metrics up in 256 character chunks. Traces of that approach can be found in TEX distributions where one such font comes with dozens of tfm files.

In order for TEX to do its work, font metrics are needed and these come, in the case of pdfTEX, from tfm files. These files contain metric information (width, height, depth and italic correction), recipes for ligatures (multiple glyphs replaced by one), and inter-character kerning data. The backend (build-in or external) will eventually filter the shapes from a font resource by index. Such a resource can be a pk file (direct index mapping), a Type1 (via an encoding vector) or a ttf file (also via an encoding vector).

A binary tfm file is either handcrafted or the product of a conversion, for instance by `afmtotfm` where the human readable afm file that can come with a TrueType font describes the font. Because TEX can handle ligatures, it is no surprise that when that information is available it will end up in the tfm file. The converter sees glyphs with names like `f`, `i` and `fi` and can add the information to the tfm file that an **f** followed by an **i** becomes **fi**.

Now, when we move on to a modern OpenType aware engine, we no longer need the tfm file but load the ttf file directly and here is where we can be surprised. The following example is in ConTEXt speak, but the principles remain:

```
\definefont [MyFont] [file:TYFATECE.TTF*default @ 45pt]
```

Here we define a font and apply the default feature set that sets up the font to provide ligatures (the `liga` feature) as well as kerns (the `kern` feature).

```
\MyFont efficient fietsen
```

efficient fietsen

When Jano Kula was working on a schedule for a film festival he wanted to use the font (based on a design by Josef Týfa) shown here, so when no ligatures showed up he was puzzled by the fact that it had worked before. That made me wonder what happened. The font is two decades old and has not changed. But what did change was the engine being used. In LuaTEX (and LuaMetaTEX) we directly load the ttf file but that particular loaded file has no features! This can be seen in the so called `tma` file in the font cache, a human readable Lua file. This is because at the time that font was made, there were no features. The ttf file was just an alternative for a Type1 file and features had to be derived from the afm file.

So, how do we deal with this? Well, the solution is similar to what these aforementioned converter commands do. First we define an additional feature, then we use it.

```
\startluacode
fonts.handlers.otf.addfeature {
    name = "myliga",
    type = "ligature",
    data = {
        ["fi"]  = { "f", "i" },
        ["fl"]  = { "f", "l" },
        ["ffi"] = { "f", "f", "i" },
        ["ffl"] = { "f", "f", "l" },
        ["fff"] = { "f", "f", "f" },
    }
}
\stopluacode

\definefontfeature[myliga][default][myliga=yes]

\definefont [MyFontLiga] [file:TYFATECE.TTF*myliga @ 45pt]

\MyFontLiga efficient fietsen
```

For the sake of this summary we used `myliga` but one can also just use `liga` as name before the first font is loaded. However, by being explicit we know what gets applied. Here is the result:

efficient fietsen

The main reason for bringing this up is that a migration to newer technology can result in (initial) unexpected loss of functionality. But fortunately in this case there is a way out. However, one can wonder if the loss of the ligatures was really that bad here. Just for the record, the following approach simulates the way TEX does it: pair-wise ligature building: work

```
\startluacode
fonts.handlers.otf.addfeature {
    name  = "myliga1",
    type  = "ligature",
    data = {
        ["fi"]  = { "f", "i" },
    }
}
fonts.handlers.otf.addfeature {
    name  = "myliga2",
    type  = "ligature",
    data = {
        ["ffi"] = { "f", "fi" },
    }
}
\stopluacode

\definefontfeature[myliga][default][myliga1=yes,myliga2=yes]
```

But the earlier solution where we just put all lookups in one feature is of course more efficient. Other solutions can use so called contextual lookups but that is overkill here.

Hans Hagen

UTF8 in MetaPost

Around the time Alan Braslau and I were discussing his new MetaFun node module, I made the MetaPost library accept utf8, if only because nowadays that is the preferred portable file encoding. Before I show what that brings us, let's see how the TEX suite deals with input.

When TEX and MetaFont, the program that MetaPost shares much of its code with, evolved, punch cards were widely used. There was quite some diversity in the word size of computers and those were not always multiples of eight. Initially characters in the engines took seven bits but soon that became eight bits. When a character was read from file, it went though a re-mapper that turned the input byte into one that was normalised inside the engine. Before something was shown to the user (on the console or in the log) there was a conversion to the preferred output encoding. Internally ascii was used, but the outer world could for instance talk ebcdic.

In the previous paragraph I talk in the past tense because in the extended TEX engines that I use, LuaTEX and LuaMetaTEX, this mapping is gone: they have a utf8 code path. In the MetaPost library that is used in LuaMetaTEX the mapping is also gone because in practice it was a one-to-one mapping, an unused leftover from the past (one can consider it an old system dependency).

The TEX and MetaPost engines differ in the way that they deal with characters. In TEX a character has a so called catcode. For instance a dollar is a math shift character (it begins or ends math mode) and its code is 3. A space has code 10, a comment 14, etc. There are 16 catcodes and if you want to know more about that: read the TEX book! It's one of these intriguing properties of TEX.

In MetaPost characters don't have catcodes but they are grouped into classes that drive the expression scanner, like left or right bracket or parenthesis. They also play a role in prioritising operators. In TEX characters with a code larger than 127 are valid and depending on how a macro package is set up they have category letter or other. In stock MetaPost they are illegal. However, in MetaPost we can make them letters (one of the classes) after which the engine will just accept them and not complain. In the wide TEX engines the characters > 127 signals a multibyte utf8 sequence, which also means that the related character code ends up as glyph reference. If you want a specific utf sequence to be a valid letter in a macro name, you need to make sure it has the right catcode for that: you need to set that up (think of Chinese with thousands of characters). In MetaPost it's easier: just put all in the characters in the 128-255 range in the letter class and you're done. One can tell the library to do that with a simple flag and we're done: MetaPost can do utf8. All it takes is this:

```
for (int k = 127; k <= 255; k++) {
    mp->char_class[k] = mp->utf8_mode
                      ? letter_class
                      : invalid_class;
}
```

After this rather trivial patch (of course one needs to set the mode) we can do the following and get figure 1:

```
\startMPcode
vardef dœn_knüth   = textext("Don Knuth")    enddef ;
vardef ДональдКнут = textext("Donald Knuth") enddef ;

draw             ДональдКнут xsized 10cm             withcolor "middlegray";
draw             dœn_knüth   xsized  4cm             withcolor "darkred";
draw             dœn_knüth   ysized  5mm rotated  45 withcolor "darkgreen";
draw textext(str dœn_knüth)   ysized  5mm rotated -45 withcolor "darkblue";
draw textext(str ДональдКнут) ysized  5mm rotated  90 withcolor "darkgray";
\stopMPcode
```

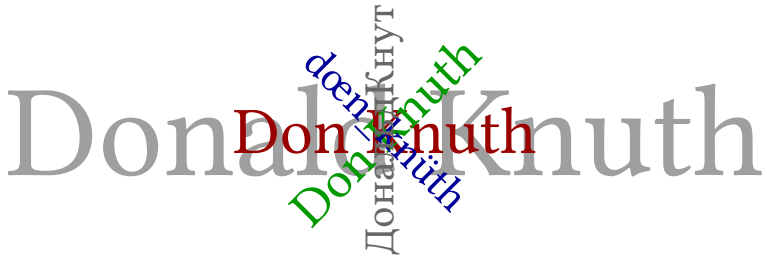


Figure 1.

But, as I mentioned that Alan and I were playing with this, a more tantalising example is possible; the result is shown in figures 2 and 3):

```
\startMPcode
  save p ; pen p ; p := currentpen ;
  pickup pencircle scaled .05;

  picture ○ ; ○ := image (draw fullcircle) ;
  picture ◎ ; ◎ := image (draw fullcircle ; draw fullcircle scaled .5) ;

  currentpen := p ;

  draw ◎ ysized 2cm withcolor "darkblue" ;
  draw ○ ysized 2cm shifted (4cm,0) withcolor "darkred" ;
\stopMPcode
```

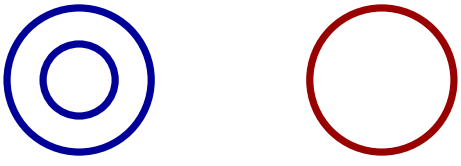


Figure 2.

```
\startMPcode
  draw image (
    for i=1 upto 100:
      draw ◎ scaled .3i shifted ((i/2)*mm,0) rotated (i*10)
             withcolor (i*red/100);
    endfor ;
  ) shifted (4cm,8cm);
\stopMPcode
```

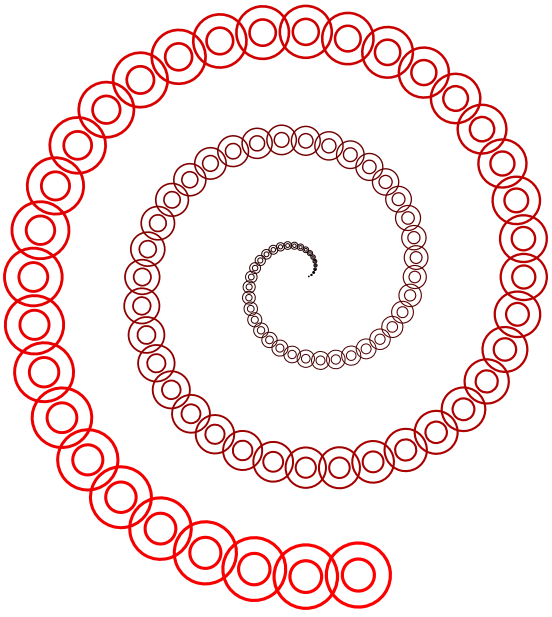


Figure 3.

In the end we came up with a bunch of symbols that can be used as indicators in graphics for tagging data points:

```
\startMPcalculation
begingroup
  pen savedpen ; savedpen := currentpen ;
  pickup pencircle scaled .05 ;
  interim ahlength := .5 ;
  interim ahvariant := 1 ;
  picture ○ ; ○ = image(draw fullcircle) ;
  picture ◎ ; ◎ = image (draw fullcircle ;draw fullcircle scaled .5) ;
  picture □ ; □ = image(draw fullsquare) ;
  picture ◇ ; ◇ = □ rotated 45 ;
  picture △ ; △ = image(draw (dir 90--dir 210--dir 330--cycle) scaled (2/3)) ;
  picture ▽ ; ▽ = △ rotated 180 ;
  picture ◁ ; ◁ = △ rotated  90 ;
  picture ▷ ; ▷ = △ rotated -90 ;
  picture ● ; ● = image(fill pathpart ○) ;
  picture ■ ; ■ = image(fill pathpart □) ;
  picture ◆ ; ◆ = image(fill pathpart ◇) ;
  picture ▲ ; ▲ = image(fill pathpart △) ;
  picture ▼ ; ▼ = image(fill pathpart ▽) ;
  picture ◀ ; ◀ = image(fill pathpart ◁) ;
  picture ▶ ; ▶ = image(fill pathpart ▷) ;
  picture ↑ ; ↑ = image(drawarrow (0,-1/2)--(0,1/2)) ;
              setbounds ↑ to unitsquare;
  picture → ; → = ↑ rotated 180;
  picture ↓ ; ↓ = ↑ rotated  90;
  picture ← ; ← = ↑ rotated -90;
  pickup savedpen ;
endgroup ;
\stopMPcalculation
```

These symbols can now be used as follows, see figure 4 for the result:

```
\startMPcode
save n ; n := 0 ;
for symbol = ○, ◎, □, ◇, △, ▽, ◁, ▷, ●, ■, ◆, ▲, ▼, ◀, ▶, ↑, →, ↓, ← :
  draw symbol scaled 4mm shifted (n, 0) ;
  n := n + 6mm ;
endfor ;
\stopMPcode
```

Figure 4.

Of course, when we have many such symbols, using a font with these characters in combination with the `textext` command is more efficient because then we just refer to a shape in a font.

The possibilities are endless. Take the following:

```
\startMPcalculation
  def ○ = fullcircle enddef ;
  def ✁ = cutafter enddef ;
  def ✂ = cutbefore enddef ;
  def ✏ = withpen pencircle enddef ;
  def ✖ = scaled enddef ;
  def ◷ = rotated - enddef ;
  def ◴ = rotated enddef ;
\stopMPcalculation
```

This might draw icon and Emoji freaks to MetaPost, but it might equally well make potential users look the other way (see figure 5):

```
\startMPcode
  draw (○ ✁ point 2 of ○) ✖ 2cm ✏ ✖ 5mm ◷ 90 withcolor "darkred" ;
  draw (○ ✂ point 2 of ○) ✖ 2cm ✏ ✖ 5mm ◴ 90 withcolor "darkblue" ;
\stopMPcode
```

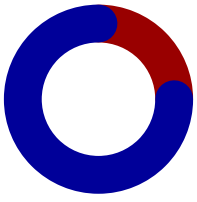


Figure 5.

But, as with many obscure features in macro packages, I'm sure that users will find a way to (ab)use this feature. Just for the record: the `textext` command is the MetaFun way to get typeset text, and using string for colours is just a convenient way to access colours at the TEX end (a redefinition of a primitive). The `MP` wrapper commands deal with runtime MetaPost processing and embedding.

Hans Hagen

Extensions related to programming macros

Introduction

Many thanks to Karl Berry who improved the English while copy-editing the following text for the spring 2021 issue of TugBoat.

Sometimes you can read (or hear) comments about TEX not being a real programming language or the wish for it to be more like a typical procedural language. A discussion about this is somewhat pointless because it relates to experiences and preferences. Also, when we mention TEX, we are talking about an interpreter, a language, a set of macros and in practice, about an ecosystem, simply because all kinds of resources are involved—especially the ecosystem is one reason why a successor is not showing up.

So, when we discuss the language aspect, it concerns a macro language and that is for a good reason: one can mix content and operations on that content in one document source. That source is interpreted and processed as it goes. This is contrary to a procedural language, where one explicitly has to push content into some procedure. These are a bit of a mix, e.g., webpage templates where some elements are snippets of programs and a preprocessor assembles the result.

```
\def\MyMacroA#1{This or #1!}
\def\MyMacroB{that}

\MyMacroA{\MyMacroB}
```

Here the last line will result in "This or that!" ending up in the output. But it must be noted that `\MyMacroB` is passed as a token, and only in the body of the macro does it get expanded into "that".

```
\edef\MyMacroC{\MyMacroB}
```

The code above defines a new macro with the expanded text as body. To expand or not, that is often the question. Now compare this code with the following:

```
function MyFunctionA(one)
    return "This or " .. one .. "!"
end
function MyFunctionB()
    return "that"
end
function MyFunctionC(one)
    return "This or " .. one() .. "!"
end

MyFunctionA("that")
MyFunctionA(MyFunctionB())
MyFunctionC(MyFunctionB)
```

The first function expects a string and returns a concatenation. The second function returns a string. The first call gets a string passed and the second one too because we call that function. But the third call passes the function itself, which is why the third function has to call it explicitly in the function body. It is this property that, in my opinion, complicates matters when you want to do typesetting in such a language: the more you nest the more dangers there are for asynchronous side effects. This can be understood from the following example:

```
function MyFunctionA(one)
    print("A")
    return "This or " .. one .. "!"
end
function MyFunctionB()
    print("B")
    return "that"
end

MyFunctionA(MyFunctionB())
```

Here we print `B` before we print `A`. Now, one can certainly argue that in spite of this, functions are easier to understand than macros (which can also have surprising side effects). Indeed, when one works on an abstract document tree where content is fetched from, say, a database that might be true but most TEX users mix content and operations.

In the following sections I will introduce some of the additional features that LuaMetaTEX provides. They are the result of experiencing many years of macro writing and the wish to come up with readable code using native features of the language when possible. Of course in ConTEXt we have a high level interface for dealing with typographical constructs and properties but deep down the code looks less clear. Putting layer upon layer doesn't help much either, so we don't go that route. Using funny characters like `!?@_:` doesn't make things look better either. We do have lots of so-called low-level macros but it doesn't make much sense to come up with a pseudo-programming layer while in fact the engine could make better facilities available; so that is the route we follow. After decades it had become clear that none of the successor TEX variants have filled in the gaps in this way, so at some point I decided that LuaMetaTEX should do it (at least for ConTEXt).

While ConTEXt MkII was written for the more traditional engines pdfTEX and XƎTEX, MkIV targets LuaTEX. It resulted in a rewrite of many components and a freeze of MkII. It made no sense to cripple ourselves so in the end we went further than originally expected. Then, when LuaMetaTEX development started, again a rewrite happened, but this time the reason was to make the code base a bit more efficient (less indirectness) by using extended native functionality. Apart from other benefits of this new engine, it gives a bit nicer code and the fewer layers we have the better. This is why ConTEXt LMTX (a.k.a. MkXL) again has a split-off code base so that MkIV is not harmed. All that said, I do admit that, lacking other TEX challenges, it is also fun to explore new venues.

Conditions

It must be said that when one goes even a little beyond simple TEX programming, one could indeed wish for a bit more comfort. Take this:[1]

```
\def\MyMacro#1#2%
  {\ifdim\dimexpr#1\relax<\dimexpr#2\relax
     less%
   \else\ifdim\dimexpr#1\relax=\dimexpr#2\relax
     equal%
   \else
     more%
   \fi\fi}
```

One needs to keep track of the nesting here in order to have the right number of `\fi`'s.

```
\def\doifelse#1#2#3#4%
  {\edef\a{#1}\edef\b{#1}%
   \ifx\a\b#3\else#4\fi}
```

The temporary macros are needed in order to be able to compare the expanded meanings. But when `#3` and `#4` are macros that look ahead you can imagine that when they see `\else` or `\fi` things can get confused. Compare this to:

```
function doifelse(a,b,c,d)
    if a == b then
        c()
    else
        d()
    end
end
```

Here the compiler creates code that calls either `c` or `d` without them having to bother about leaving the condition. In TEX-speak we would need to have something like this:

```
\def\firstoftwoarguments #1#2{#1}
\def\secondoftwoarguments#1#2{#2}
\def\doifelse#1#2#3#4%
  {\edef\a{#1}\edef\b{#1}%
   \ifx\a\b
     \expandafter\firstoftwoarguments
   \else
     \expandafter\secondoftwoarguments
   \fi}
```

And when you try that with the first example where we had a nested condition you can imagine that it quickly starts looking complex. Another aspect of the last macro is that it uses two temporary macros that better have names that don't clash, so the ones we choose here are pretty bad. I will come back to dealing with that later.

One gets accustomed to this and often this kind of code is hidden from the user so only macro writers are victims here. But, being one myself, the question is, can we make the code look nicer?

1. We use a `\dimexpr` because we cannot use a terminal percentage or space if we want to be fully expandable and don't want spaces to creep in after one token arguments.

Let's redo the first example with LuaMetaTEX:

```
\def\MyMacro#1#2%
  {\ifdim\dimexpr#1\relax<\dimexpr#2\relax
     less%
   \orelse\ifdim\dimexpr#1\relax=\dimexpr#2\relax
     equal%
   \else
     more%
   \fi}
```

Many programming languages have something like `elseif` but because TEX has quite a number of different tests, `\elseifdim` makes no sense but the more generic `\orelse` does. We can even think of:

```
\def\MyMacro#1#2%
  {\ifcmpdim\dimexpr#1\relax\dimexpr#2\relax
     less%
   \or
     equal%
   \else
     more%
   \fi}
```

And because LuaMetaTEX provides this test, one obstacle is gone. We leave it to the reader to come up with a traditional TEX implementation of this:

```
\def\MyMacro#1#2%
  {\ifcmpdim\dimexpr#1\relax\dimexpr#2\relax
     \expandafter\firstofthreearguments
   \or
     \expandafter\secondofthreearguments
   \else
     \expandafter\thirdofthreearguments
   \fi}
```

And how nice it would be to be able to do this:

```
\def\doifelse#1#2%
  {\iftok{#1}{#2}%
     \expandafter\firstoftwoarguments
   \else
     \expandafter\secondoftwoarguments
   \fi}
```

And so, LuaMetaTEX has such a primitive test. Keep in mind that defining `\iftok` as a macro is possible here but that won't work well nested, even with `\orelse`:

```
\iftok{.}{.}
\orelse\iftok{..}{..}
\orelse\iftok{...}{...}
\fi
```

When a condition succeeds or fails TEX enters fast scanning mode to skip over the branch that is not used. For that it needs to know if a token is a test, which is why defining `\iftok` as a macro is no help. We could flag a macro as a test and I actually played with this, but it means that we need to test a macro property independent of the current condition handler and that is something for later. As an intermediate solution we have an `\ifcondition` primitive that is seen as a condition when fast

scanning happens and as a no-op when a condition is expected in which case the following macro has to expand to a condition itself. Something like this:

```
\ifcondition\mytest{.}{.}
\orelse\ifcondition\mytest{..}{..}
\orelse\ifcondition\mytest{...}{...}
\fi
```

Because we have Lua there are also ways to let Lua functions behave like if tests but that is beyond this overview, since it goes beyond the macro language. In ConTEXt we use this feature to implement some bitwise operations and tests.

In the engine we provide this repertoire of tests: `\if`, `\ifcat`, `\ifnum`, `\ifdim`, `\ifodd`, `\ifvmode`, `\ifhmode`, `\ifmmode`, `\ifinner`, `\ifvoid`, `\ifhbox`, `\ifvbox`, `\ifx`, `\iftrue`, `\iffalse`, `\ifcase`, `\ifdefined`, `\ifcsname`, `\iffontchar`, `\ifincsname`, `\ifabsnum`, `\ifabsdim`, `\ifchknum`, `\ifchkdim`, `\ifcmpnum`, `\ifcmpdim`, `\ifnumval`, `\ifdimval`, `\iftok`, `\ifcstok`, `\ifcondition`, `\ifflags`, `\ifempty`, `\ifrelax`, `\ifboolean`, `\ifmathparameter`, `\ifmathstyle`, `\ifarguments`, `\ifparameters`, `\ifparameter`, `\ifhastok`, `\ifhastoks` and `\ifhasxtoks`.

Some of these are variants of `\ifcase`, needed when there are more than two outcomes possible. In addition there are `\unless`, `\else`, `\or`, `\orelse` and `\orunless`. The new primitives are discussed in documents that come with the ConTEXt distribution.

With respect to testing arguments, you can also use the pseudo-counter `\lastarguments` (watch the 'last' in the name) and somewhat less efficient but more reliable `\parametercount` instead as these are indicators of the number of passed commands.

Protection

In the previous section we mentioned that using auxiliary macros is tricky because they can clash with existing macros. In fact, this is true for any macro! I therefore decided to do what has been on the agenda for a while: add a mechanism that protects against overload. This is still experimental and the impact on users can only be tested after most ConTEXt users have switched to LMTX, which may take a while. This also means that it will take a while before the related primitives are considered stable (although I'm sure not much will change). Let's take a previous example:

```
\permanent\def\firstoftwoarguments #1#2{#1}
\permanent\def\secondoftwoarguments#1#2{#2}
\permanent\protected\def\doifelse#1#2%
  {\iftok{#1}{#2}%
     \expandafter\firstoftwoarguments
   \else
     \expandafter\secondoftwoarguments
   \fi}
```

Here the three macros are defined as permanent. The test itself is protected against expansion (which it has always been so we keep that). Depending on the value of the `\overloadmode` variable (discussed below) a user will get a warning or fatal error. By default there is no checking (but I might give the `\immutable` prefix, also discussed below, an "always check for it" property).

The whole repertoire of prefixes related to overload protection is given in the following table.

`frozen`	a macro that has to be redefined in a managed way
`permanent`	a macro that had better not be redefined
`primitive`	a primitive that normally will not be adapted
`immutable`	a macro or quantity that cannot be changed, it is a constant
`mutable`	a macro that can be changed no matter how well protected it is
`instance`	a macro marked (for instance) to be generated by the user interface
`overloaded`	when permitted the flags will be adapted
`enforced`	all is permitted (but only in zero mode or 'initex' mode)
`aliased`	the macro gets the same flags as the original

For the first five the primitive state has no related prefix primitive; it is set by the engine itself. Maybe someday I will decide to permit defining primitives, which would take hardly any code to implement. Permanent macros are (as shown) those that we don't want users to redefine, and frozen ones are mildly protected. They can be redefined when the `\overloaded` prefix is used. A mutable macro can always be redefined, think of temporary macros, while an immutable can never be redefined. The instance property is just a signal that we're dealing with an instance, which can be handy when we trace. The `\aliased` prefix will copy properties, so this:

```
\aliased\let\forgetaboutit\relax
```

makes `\forgetaboutit` a reference to the current meaning of `\relax` (because that is what `\let` does) but also protects it like a primitive (because that is what `\relax` is).

The `\enforced` prefix is special. It only has a meaning inside a macro body or token register and it gets converted in a (hidden) `\always` prefix when in so-called ini mode (when the format is made). This permits system macros to overload in spite of heavy protection against it. Think of macros like `\NC` where the meaning can differ depending on the kind of table mechanism used, or `\item` which can differ by environment. We can protect these against overloading by the user but still redefine them. Of course, when the overload mode is zero, all can be redefined.

The value of `\overloadmode` determines to what extent a user will be annoyed when an existing macro is redefined, as shown in the table below. That can also be an instance defined by commands like `\definehighlight` although these normally are just `\frozen \instance` which means that a low level of protection only issues a warning.

		immutable	**permanent**	**primitive**	**frozen**	**instance**
1	warning	⋆	⋆	⋆		
2	error	⋆	⋆	⋆		
3	warning	⋆	⋆	⋆	⋆	
4	error	⋆	⋆	⋆	⋆	
5	warning	⋆	⋆	⋆	⋆	⋆
6	error	⋆	⋆	⋆	⋆	⋆

The even values (except zero) will abort the run. A value of 255 will freeze this parameter. At level five and above the instance flag is also checked but no drastic action takes place. We use this to signal to the user that a specific instance is redefined (of course the definition macros can check for that too).

Alignments

In ConTEXt many commands are defined using the prefix \protected, which is handy when they are used in a context where expansion would not work out well, like writing to file or inside an \edef. However, this is impossible when we use the alignment mechanism. This has to do with the fact that the parser looks ahead to see if we have (for instance) a \noalign primitive. And since the parser doesn't look inside a \protected macro, this fails:

```
\protected\def\MyMacro{\noalign{\vskip 10pt}}
```

It also works out badly for macros that look for arguments. A dirty trick is:

```
\def\MyMacroA{\noalign\bgroup\MyMacroB}
\def\MyMacroB{\dosingleempty\MyMacroC}
\def\MyMacroC[#1]{....\egroup}
```

This somewhat over the top approach can now (in LuaMetaTEX) be simplified to the following. Let's also go crazy with prefixes here:

```
\noaligned\permanent\tolerant\protected\def\MyMacroA[#1]%
  {\noalign\bgroup....\egroup}
```

For the record: in LuaMetaTEX the \noalign construct can be nested which again simplifies some (ConTEXt) code. Keep in mind that until now we could do whatever we wanted in traditional TEX speak, apart from making such macros \protected.

Definitions

From the perspective of the above it will become clear that in a system like ConTEXt quite a number of definitions are candidates for being flagged. You also need to think of symbolic character names or math symbols. For instance dimensions defined by \dimendef also get a permanent status. This means that one cannot redefine \scratchcounter but still its value can be changed. At this moment I see no reason to have a flag for preventing that (also because it would add overhead), but it might become an option some day.

However, there are often quantities that need overload protection, such as constant values. This is why we have:

```
\immutable \integerdef    \plusone   1
\immutable \dimensiondef  \onepoint  1pt
\immutable \gluespecdef   \zeroskip  0pt plus 0pt minus 0pt
\immutable \mugluespecdef \onemuskip 1mu
```

Those will never change and are a macro-like variant of registers but with an efficient storage model and behaving like a register. But one cannot use the operators like \advance on them. Their intended usage is as a constant.

Another definition-related extension involves \csname. In LuaTEX we introduced more robust handling of \ifcsname as well as an extra accessor:

```
\ifcsname f o o\endcsname
    \lastnamedcs % reference to the constructed \cs
\fi
```

as well as:

```
\begincsname f o o\endcsname
```

which doesn't define \f o o as a 'relaxed' macro when it doesn't already exist. Both \begincsname and \lastnamedcs avoid a second name construction, as in:

```
\ifcsname f o o\endcsname
    \csname f o o\endcsname
\fi
```

Keep in mind that these additions are a side effect of control sequences being in utf-8 format so we want to avoid unnecessary construction of temporary strings and related expansion.

Original TEX only has `\csname`; ε-TEX and LuaTEX added some companion primitives to that, and LuaMetaTEX again extends the repertoire:

```
\letcsname  f o o\endcsname\relax
\defcsname  f o o\endcsname{...}
\edefcsname f o o\endcsname{...}
\gdefcsname f o o\endcsname{...}
\xdefcsname f o o\endcsname{...}
```

This saves passing some arguments to a helper like `\setvalue` which is a bit more efficient and it also saves a token. (The ConTEXt format file became quite a bit smaller when the extensions discussed here were applied.) The `\ifcsname` primitive has been made somewhat more efficient by honoring macros that were defined as `\protected` which (we think) means: don't expand me in those cases where it makes no sense. So here we have an (in my opinion) acceptable downward incompatibility with engines that conform to ε-TEX.

There are a few more definition related new primitives, like:

```
\glet\MyMacroA\MyMacroB            % shortcut for \global\let
\swapcsvalues\MyMacroA\MyMacroB    % also works for registers
\futuredef\DoWhatever\MyMacro{...}
\expand\MyProtectedMacro           % \protected like this
```

Arguments

Let's start with a teaser. A previous definition needed a helper to gobble one of two arguments. The following does the same but it just gobbles and doesn't store the argument, which is why we use `#1` in both cases. This avoids storing token lists for the unused arguments.

```
\permanent\def\firstoftwoarguments #1#-{#1}
\permanent\def\secondoftwoarguments#-#1{#1}
```

Because anything other than a digit after a # triggers an error I saw no reason not to support some more: it doesn't hurt downward compatibility, unless you use TEX to generate error messages. Here is the full list of extensions, of which I will discuss a few (more can be found in the ConTEXt distribution and source code).

`+`	keep the braces
`-`	discard and don't count the argument
`/`	remove leading and trailing spaces and pars
`=`	braces are mandatory
`_`	braces are mandatory and kept
`^`	keep leading spaces
`1-9`	an argument
`0`	discard but count the argument
`*`	ignore spaces
`:`	pick up scanning here
`;`	quit scanning

We have a few useful characters left, such as < and > so who knows what future extensions might show up.

Delimited arguments are used frequently in ConTEXt; take this:

```
\def\MyMacro[#1][#2]{...}
```

Here the call is rather sensitive, for instance this will fail:

```
\MyMacro[A] [B]
```

We can cheat and define:

```
\def\MyMacro[#1]#2[#3]{...}
```

in which case #2 gets what sits between the brackets. But still these two arguments have to be given. So, in MkII and MkIV you will find indirectness like the following:

```
\def\MyMacro{\dodoubleempty\doMyMacro}
\def\doMyMacro[#1][#2]{}
```

However, in LMTX you can find this alternative:

```
\tolerant\def\MyMacro[#1]#*[#2]{...}
```

The \tolerant will make the parser quit when no match can be made and the #* will gobble spaces. In fact, we often do this:

```
\tolerant\protected\def\MyMacro[#1]#*[#2]{...}
```

and if we want overload protection:

```
\permanent\tolerant\protected\def\MyMacro[#1]#*[#2]{...}
```

The combination of \tolerant and \protected with either expansion or not of a macro gives four variants of low-level macro commands: *normal*, *tolerant normal*, *protected* and *tolerant protected*. In LuaTEX that protection against expansion is implemented in a more indirect way, just like in ε-TEX. There we also have \long and \outer properties so we have *normal*, *long normal*, *outer normal* and *long outer normal*. Making protected against expansion a native command would have given another four command codes. Combining that with tolerant would again double it so we then would end up with 16 command codes. But in LuaMetaTEX we dropped the long and outer properties. In ConTEXt we never used outer and always want long anyway.

The reason for mentioning these details is to make clear that the introduced overhead can be neglected when we compare to LuaTEX, apart from the fact that we gain from the expansion protection being a first class feature now, macros without arguments being stored more efficiently, the parser being a little optimized and so on.

But of course the biggest benefit is that, when we look at the example above, we avoid indirectness. It looks nicer. It gives less clutter in tracing. It takes fewer tokens in the format (where each token takes eight bytes). It runs a little faster. It demands no trickery. Take your choice. For the record: you don't want to know what the set of \dodoubleempty macros looks like, as they themselves use indirectness and are highly optimized for performance.

The list of possible features has more than skipping spaces. Here's another example:

```
\tolerant\def\MyMacro[#1]#;(#2){<#1#2>}
```

Here \MyMacro accepts [A] and then quits or when not seen, checks for (A) and when not found is still happy. So, either #1 or #2 has a value. How do we know what arguments got grabbed? There are several ways to find out:

```
\tolerant\def\MyMacro[#1]#;(#2)%
  {\ifarguments
     % zero arguments
   \or
     % one argument
   \else
     % two arguments
   \fi}
```

This test uses the count from the last expansion so if any macro expansion happens before the test you can get the wrong value! The next test provides feedback about what argument got a value:

```
\tolerant\def\MyMacro[#1]#;(#2)%
  {\ifparameters
     % all empty
   \or
     % first has value
   \else
     % second has value
   \fi}
```

But still may not be enough so we can also explicitly test for a parameter. But again be aware of nesting:

```
\tolerant\def\MyMacro[#1]#;(#2)%
  {\ifparameter#1\or
     % first has value
   \fi
   \ifparameter#2\or
     % second has value
   \fi}
```

This is pretty robust but expands the arguments in the test:

```
\tolerant\def\MyMacro[#1]#;(#2)%
  {\unless\iftok{#1}{}%
     % first has value
   \fi
   \unless\iftok{#2}{}%
     % second has value
   \fi}
```

When we use a colon instead of a semicolon the parser knows where to pick up after a match fails:

```
\tolerant\def\MyMacro[#1]#:#2{...}
```

So, the argument between brackets is optional and the single token or braced second argument (turned into a token list) is mandatory.

The other extensions more or less speak for themselves: they grab arguments and discard or keep braces and such in cases where TEX would treat them specially when storing or passing them on. Speaking of braces, in spite of what one might expect (assuming that braces are more a TEX thing than brackets) the following two definitions perform equally well

```
\def\foo[#1]{} \foo[1]
\def\foo  #1{} \foo{1}
```

but:

```
\def\oof[#1]{}
\def\foo{\dosingleempty\oof}
```

performs more that 5 times worse than this:

```
\tolerant\def\foo[#1]{}
```

So, the added overhead (and there is some, also because we keep track of more) in the argument parser gets compensated well by the fact that we can avoid indirectness. The impact on an average document probably goes unnoticed.

As with much in TEX you need to be aware of (intentional) side effects. Take for instance:

```
\tolerant\def\foo#1[#2]#*[#3]{\edef\ofo{#1}}
\def\oof{\foo{oeps}}
```

That will probably not do what you expect. It has to do with how TEX interprets spaces in the context of argument parsing: they can become part of the argument (here #1) so anything before the first seen left bracket becomes the argument's value.

```
\tolerant\def\foo#1#*[#2]#*[#3]{\edef\ofo{#1}}
\def\oof{\foo{oeps}}
```

This however works because the first #* directive stops scanning for the first argument and then gobbles spaces when seen before continuing to look for the bracketed arguments. So TEX's charm is still there.

Introspection

Because macros have more properties and variation in arguments the `\meaning` command has a companion `\meaningfull` that displays what prefixes were applied. The `\meaningless` variant only shows the body.

Quite some effort went into normalizing the so-called command codes. Primitives are grouped into categories with similar treatments in order to keep the main loop efficient. These codes also determine the expansion contexts (think of usage in an `\edef`, how they get serialized (for instance in messages), etc. The char codes (called such because in most cases tokens represent characters of some kind) distinguish commands in these groups. Think of `\def` and `\edef` being call commands with a different code. This rather intrusive (internal) regrouping of primitives was needed in order to get a more consistent Lua token interface. So, for instance the codes are now in consecutive ranges, registers are split into internal and user variants, etc.

Also, memory management has been overhauled so we have a more dynamic allocation of various data structures (stacks, equivalents, tokens, nodes, etc.) and we use the whole 64 bit memory word to save some memory in places too. All this is the reason why it is unlikely that much will get backported to LuaTEX, also because in ConTEXt we now have a special version for LuaMetaTEX: LMTX.

There is more

Here we've discussed only the primitives that make the source look better while also being convenient. But it is worth mentioning that there are primitives like `\toksapp` and `\etokspre` that append and prepend tokens to a register (there are eight variants). There are ways to collect tokens for just before or after a group ends. There are some new expansion related primitives like `\expandtoken` that can be used to inject a token with some specific catcode, just like one can define active characters without the need for dirty uppercase tricks.

The typesetting department also has extensions. We can freeze paragraph properties, adjust math parameters locally, normalize lines so that at the Lua end we know

what to expect (think of consistent presence of left and right skip, left and right shape related properties, left and right parfill skips, indentation being glue, etc.). Hyphenation can be controlled in more detail too, and left and right side ligatures and kerns can be influenced in the running text and go with glyphs. Talking of glyphs, there are advanced scaling options as well as support for influencing placement in the running text, which permits more efficient font handling. Boxes have more properties too: they can have offsets, an orientation, etc. which makes implementing vertical typesetting a bit easier. Rules also have shifts. We can register actions to be expanded at the end of a paragraph. All this evolved over time and has been tested in ConTEXt but will be applied more frequently after the complete code split between MkIV and LMTX. That process goes hand in hand with adapting to the new situation, remove old (obsolete) variants, removing still present experimental code, etc.

There is more but hopefully this gives an impression of how substantial the LuaMetaTEX engine differs (in added functionality) with its ancestors. Maybe it looks a bit over the top, but I did actually reject some ideas after experimenting with them. On the other hand there are still some on the agenda. For instance the engine can migrate and carry around so-called deeply buried inserts pretty well now but dealing with inserts could be made a bit easier (think of columns). So, we're not done yet.

It should be noted that contrary to what one might expect the code base is still quite okay and the binary stays well below 3 MB. In the meantime memory management is also improved and the format file got smaller. A lot of the internal reorganization relates to the fact that we have a Lua interface and exposing internals demands consistency, avoidance of (often clever) tricks, more abstraction, etc.

It is also worth noting that we can only do such a massive operation because users are willing to test intermediate versions (sometimes on very large projects) and because all changes in the code base are meticulously checked by Wolfgang Schuster who knows TEX and ConTEXt inside out. And of course we have Mojca Miklavec's compile farm to keep it available for all relevant platforms, where we use a mix of `gcc` (also with cross compilation), `clang` and `msvc` for various platforms, up to date. It definitely helps that compilation is fast (due to the refactored code base) and that I can use Visual Studio to work with the code.

In this summary I only covered some aspects of TEX. Another important set of extensions concerns the MetaPost library, where token scanners are exposed, more advanced Lua calls are possible and where no longer relevant bits of code have been removed. And we use the latest and greatest Lua 5.4—but discussing the implications of these is for another article.

Hans Hagen

Playing with Axodraw

Abstract
This paper shows some of the features of Axodraw. It puts emphasis on applications, not only in the field of physics, but also in completely unrelated fields like mathematical tiling constructions, fashion patterns or the design of sudokus.

Introduction

Question: what do the figures 1–5 have in common?
The answer to the above question is: all were made with the help of Axodraw.

A first version of Axodraw was created in 1992. At the time LaTeX did not have any sophisticated graphical capabilities beyond lines and the creation of fonts that might contain half circles and line segments. Attempts at drawing Feynman diagrams resulted either in very ugly graphics or the inclusion of an external file for each individual diagram/picture which had to be made by a separate drawing package. The advent of the NeXT computer with a screen that ran display postscript, combined with the possibilities to include postscript commands in TeX and LaTeX

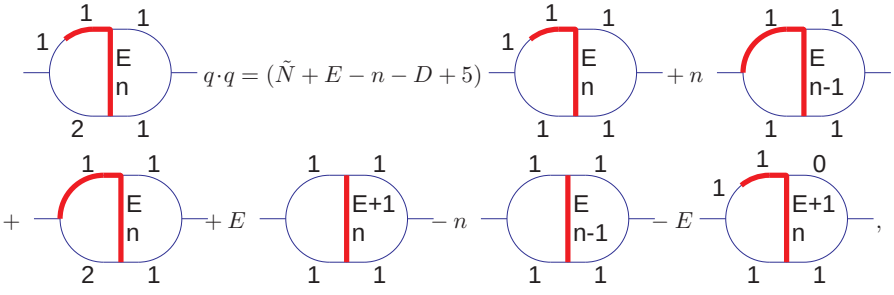


Figure 1.

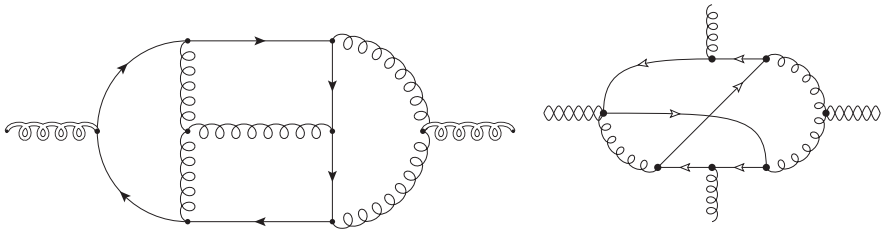


Figure 2.

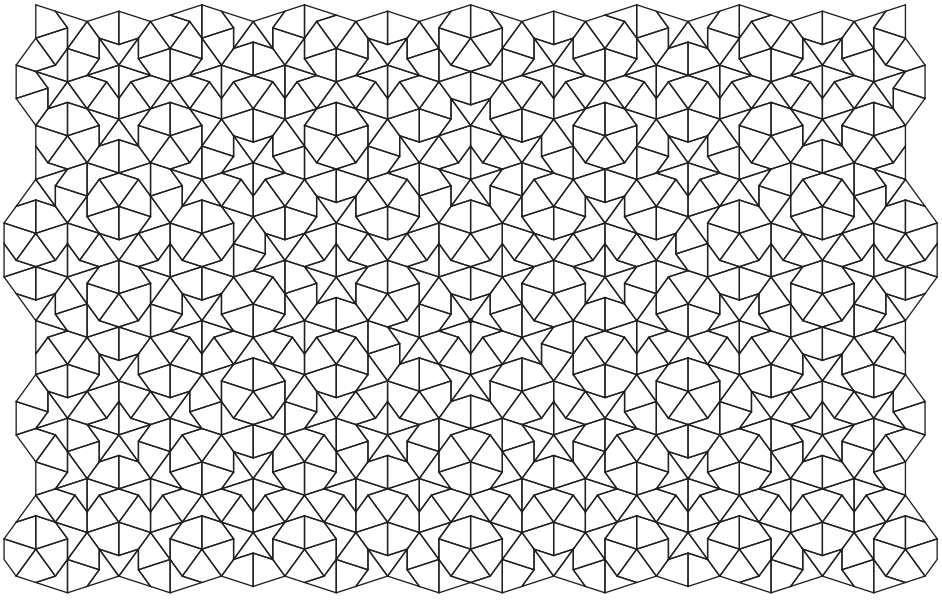


Figure 3.

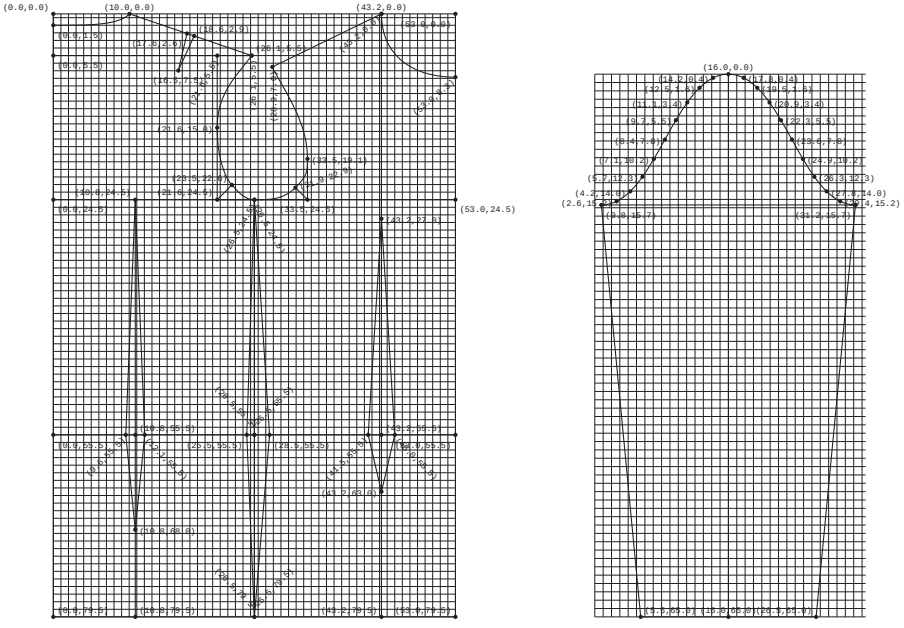


Figure 4.

files made it possible to design a style file that contained many graphical primitives inside the LaTEX picture environment. This gave much better graphics and also allowed all diagrams to be part of the LaTEX source file. At first this was for private use, but on demand it was made publicly available and published in CPC [1].

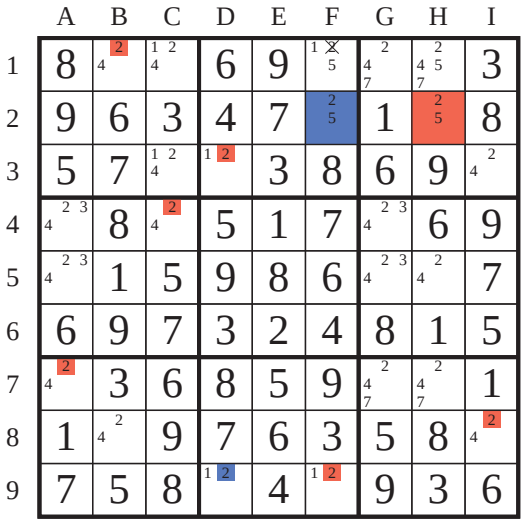


Figure 5.

Version 1

Version 1 worked with the use of postscript specials. Such specials pass the code inside them on to the dvips program that converts the dvi code to postscript. Hence the flow of translation is

```
latex file
dvips file -o
```

and the result would be file.ps with the postscript code inside the specials now inside the .ps file. Here is an example of such a special

```
\special{! /bbox{
%
%   Draws a blanked out box x1,y1,x2,y2
%
    gsw p2 p1
    gsave
    1 setgray abox fill
    grestore
    abox stroke
    grestore
} def }
```

and the LaTeX interface for this function:

```
\def\BBox(#1,#2)(#3,#4){
%
%   Draws a box with the left bottom at (x1,y1) and the right top
%   at (x2,y2). The box is blanked out.
%
\put(\axoxoff,\axoyoff){\special{"\axocolor #1 \axoxo add #2 \axoyo
 add #3 \axoxo add #4 \axoyo add \axowidth \axoscale bbox showpage}}
}
```

The variables here are offsets, color, linewidth and a scale factor. All postscript functions are in a preamble which, together with the LaTeX interfaces, are inside the file Axodraw.sty.

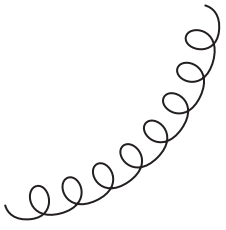
There are of course much more complicated functions, like a gluon on a circle segment:

```
\special{! /gluearc{
%
%  Draws a gluon on an arcsegment
%  x_center,y_center,radius,stat_angle,end_angle,gluon_radius,num
%  in which num is the number of windings of the gluon.
%  Method:
%  1:  compute length of arc.
%  2:  generate gluon in x and y as if the arc is a straight line
%  3:  x' = (radius+y)*cos(x*const)
%      y' = (radius+y)*sin(x*const)
%
   gsw /num ed /ampi ed /arcend ed /arcstart ed /radius ed
%
%  When arcend comes before arcstart we have a problem. The solution
%  is to flip the order and change the sign on ampi
%
   arcend arcstart lt {
       /ampi ampi -1 mul def
       /arcstart arcend /arcend arcstart def def
   } if
%
   translate                              % move to center of circle
   arcstart rotate                          % segment starts at zero
   /darc arcend arcstart sub def             % argsegment
%
  /dr darc 180 div 3.141592 mul radius mul def  % length of segment.
%
   /const darc dr div def                    % conversion constant
%
   /num num 0.5 sub round def
  /inc dr num 2 mul 2 add div def       % increment per half winding
%
   /amp8 ampi 0.9 mul def
   /amp1 radius ampi add def
   /amp2 radius ampi sub def
   /amp3 radius ampi 2 div add def
   /amp4 amp1 inc amp8 add const mul cos div def
   /amp5 amp2 amp8 const mul cos div def
   /amp6 amp1 inc 0.6 mul amp8 add const mul cos div def
   /amp7 amp1 inc 0.9 mul const mul cos div def
   amp8 0 lt {/amp8 amp8 neg def} if
%
   /x1 inc 2 mul def
%
   newpath
       radius 0 moveto
%
       inc 0.1 mul const mul dup cos amp3 mul exch sin amp3 mul
       inc 0.5 mul const mul dup cos amp7 mul exch sin amp7 mul
       inc 1.4 mul const mul dup cos amp1 mul exch sin amp1 mul
           curveto
       x1 amp8 add const mul dup cos amp6 mul exch sin amp6 mul
```

```
       x1 amp8 add const mul dup cos amp5 mul exch sin amp5 mul
       x1 const mul dup cos amp2 mul exch sin amp2 mul
           curveto
%
       2 1 num {
           pop
          x1 amp8 sub const mul dup cos amp5 mul exch sin amp5 mul
          x1 amp8 sub const mul dup cos amp4 mul exch sin amp4 mul
           x1 inc add const mul dup cos amp1 mul exch sin amp1 mul
               curveto
           /x1 x1 inc dup add add def
          x1 amp8 add const mul dup cos amp4 mul exch sin amp4 mul
          x1 amp8 add const mul dup cos amp5 mul exch sin amp5 mul
           x1 const mul dup cos amp2 mul exch sin amp2 mul
               curveto
       } for
%
       x1 amp8 sub const mul dup cos amp5 mul exch sin amp5 mul
       x1 amp8 sub const mul dup cos amp6 mul exch sin amp6 mul
      x1 inc 0.6 mul add const mul dup cos amp1 mul exch sin amp1 mul
           curveto
      x1 inc 1.5 mul add const mul dup cos amp7 mul exch sin amp7 mul
      dr inc 0.1 mul sub const mul dup cos amp3 mul exch sin amp3 mul
       dr const mul dup cos radius mul exch sin radius mul
       curveto
   stroke
%
   grestore
} def }
```

Its use is with:

```
\begin{center}
\SetScale{1.5}
\begin{axopicture}(70,70)
\GluonArc(10,60)(50,270,360){4}{8}
\end{axopicture}
\end{center}
```



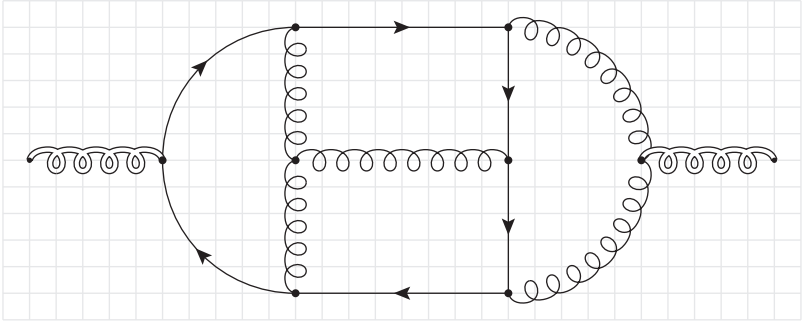
Version 2

A big weakness of Axodraw was that one needed to figure out the coordinates. D.Binosi en L.Theussl solved that problem by creating an interactive Java program named JAxodraw [2] in which one can select graphical elements and position them with the mouse. When the figure is complete one can let it generate the proper Axodraw code. It is still used very much.

When John Collins was writing a book on field theory, he needed more graphical primitives. At first this resulted in a second version of JAxodraw [3], but in the end it was not sufficient.

Eventually John and I have made a second version of Axodraw in which many of the user remarks were addressed. For instance the coordinate problem was greatly improved by a grid function which is used during the design of a picture and commented out when the picture is complete.

```
\begin{center}
\begin{axopicture}(300,120)
\AxoGrid(0,0)(10,10)(30,12){LightGray}{0.5}
\Line[arrow](110,110)(190,110)
\Line[arrow](190,10)(110,10)
\Arc[arrow,clockwise](110,60)(50,270,180)
\Arc[arrow,clockwise](110,60)(50,180,90)
\GluonArc(190,60)(50,270,360){4}{8}
\GluonArc(190,60)(50,0,90){4}{8}
\Gluon(110,110)(110,60){-4}{5}
\Gluon(110,60)(110,10){-4}{5}
\Gluon[double](10,60)(60,60){4}{4}
\Gluon[double](240,60)(290,60){4}{4}
\Line[arrow](190,110)(190,60)
\Line[arrow](190,60)(190,10)
\Gluon(110,60)(190,60){4}{8}
\Vertex(110,110){1.5}
\Vertex(110,10){1.5}
\Vertex(60,60){1.5}
\Vertex(240,60){1.5}
\Vertex(190,110){1.5}
\Vertex(190,10){1.5}
\Vertex(110,60){1.5}
\Vertex(190,60){1.5}
\Vertex(10,60){1}
\Vertex(290,60){1}
\end{axopicture}
\end{center}
```



One of the wishes from my side was the capability to use pdflatex because nowadays the pdf file format is used much more than the postscript format. This is far from trivial, because pdf is not a language in which one can compute things and in addition its manual is about 1000 pages. The eventual solution is a separate program, axohelp, that generates the pdf instructions. It is used in a way that is similar to the use of makeindex:

```
pdflatex file
axohelp file
pdflatex file
```

The first run of pdflatex generates a file file.ax1 which contains a list of all Axodraw instructions that need to be translated. The running of axohelp creates a file file.ax2 that contains the original file.ax1 statements and their pdf translation. When pdflatex is run again, it sees the .ax2 file and as long as all Axodraw statements agree with the statements in the .ax2 file pdflatex can use the translation. If there is disagreement, because the input pictures have been changed, there will be the advise to run axohelp again. Hence, when no pictures have been changed and there is already a .ax2 file one can suffice with running pdflatex just once.

Hence, if in a file.ax1 the graphical object 287 would look like

```
[287] Polygon "(-163.4721,-118.7694)(-139.6218,-126.5188)
           (-148.7318,-113.9800)(-148.7318,-98.4812)" 1.000000;
```

in file.ax2 it would become:

```
\axo@setObject{287}%
{Polygon "(-163.4721,-118.7694)(-139.6218,-126.5188)
         (-148.7318,-113.9800)(-148.7318,-98.4812)" 1.000000;}%
{ 1 w -163.472 -118.769 m -139.622 -126.519 l -148.732 -113.98 l
  -148.732 -98.481 l h S}
```

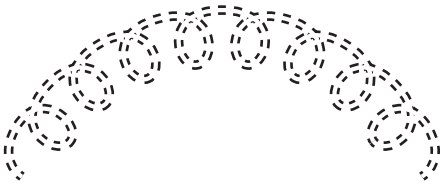
Among the new features are Bezier curves and customizable arrows that can be put at a percentage of the length of a line. This gives interesting problems like how to put a properly alligned arrow on a Bezier curve at a given fraction of its length.


```
\Bezier[arrow,arrowpos=0.20](10,10)(20,60)(80,10)(90,40)
```

To do this well you need to:

1. Compute the length of a Bezier curve.
2. Compute the length of part of a Bezier curve.
3. Iterate until you are at the proper percentage.
4. Compute the derivative at the given point.

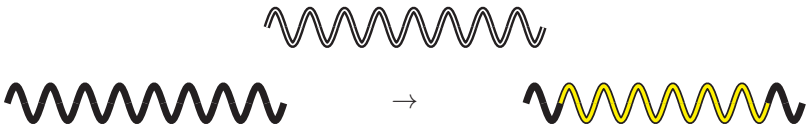
Another little problem: How to draw a double line or spiral?



```
\begin{center}
\SetScale{2.0}
\begin{axopicture})100,50)(0,0)
\DashDoubleGluonArc(45,0)(40,20,160){5}{8}{1.3}{1.5}
\end{axopicture})100,50)(0,0)
\end{center}
```

Some of these features take quite some calculations. This is easy in the C sources of axohelp, but it is also possible in postscript. In LaTeX and in pdf files this is not possible.

In general you don't draw a double line by drawing two lines, shifted from each other as in JAxodraw, but by first drawing a fat line in the foreground color and then a thinner line in the background color on top:



```
\begin{center}
\SetScale{1.3}
\begin{axopicture}(250,15)(0,0)
\Photon[width=2](10,8)(90,8){5}{8}
\Text(125,8)[]{$\rightarrow$}
\Photon[width=2](160,8)(240,8){5}{8}
\SetColor{Yellow}
\Photon[width=1](170,8)(230,8){5}{6}
\end{axopicture}
\end{center}
```

Another problem in Axodraw is how to draw a sine wave, a spiral or, in pdf, a circle? It can be approximated by a very large number of very small lines, but this is incredibly slow, specially back in 1992. The solution is to use approximations in terms of Bezier curves which are present both in postscript and pdf. A sine wave is drawn with the use of 180 degree segments (from 90 to 270 degrees and from 270 to 90 degrees) with special segments for the endpoints. If this is done well, you need to magnify it very much to see the difference with a proper sine wave. This can also be done with a spiral. The gluon has special endpoints to keep everything properly centered. This is a feature that shows immediately whether a picture has been made with Axodraw.

Postscript has circles, but pdf does not. Hence also circles and circle segments have to be approximated with Bezier curves.

The postscript output and the pdflatex output are, to my knowledge, identical, except for one detail:

In postscript there are real postscript fonts and those do not exist as such in the pdf version. There do exist LaTeX versions of these fonts, and those are used in the pdflatex version, but there is a problem with that the zero height is interpreted differently in postscript than in LaTeX and pdf. Because of this there can be a small vertical displacement between the two versions. A good example is the character q. In Postscript the zero is at the bottom.

About axohelp: the program is set up in in a way that makes it relatively easy to prepare files for other formats if those would become popular in the future. The language in which it is written is C, because this is available on all computers, and also because I am rather familiar with it. If the project would have to be redone, the language Rust might be a consideration, due to its draconic error checking.

An example of some code in axohelp:

```
static double BzK;

void BezierCircle(double r,char *action)
{
    outpos += sprintf(outpos,
        " %12.3f 0 m %12.3f %12.3f %12.3f %12.3f 0 %12.3f c\n",
        -r,-r,r*BzK,-r*BzK,r,r);
    outpos += sprintf(outpos,
        " %12.3f %12.3f %12.3f %12.3f %12.3f 0 c\n",
        r*BzK,r,r,r*BzK,r);
    outpos += sprintf(outpos,
        " %12.3f %12.3f %12.3f %12.3f 0 %12.3f c\n",
        r,-r*BzK,r*BzK,-r,-r);
    outpos += sprintf(outpos,
        " %12.3f %12.3f %12.3f %12.3f %12.3f 0 c %s\n",
        -r*BzK,-r,-r,-r*BzK,-r,action);
}
```

with $\mathrm{BzK} = \frac{4}{3}(\sqrt{2}-1)$, or

```
void Polygon(double *args,int num,int type)
{
    int i;
    MoveTo(args[0],args[1]);
    args += 2;
    for ( i = 1; i < num; i++, args += 2 ) {
        LineTo(args[0],args[1]);
    }
    if ( type == 0 ) { CloseAndStroke; }
    else if ( type ==  1 ) { CloseAndFill; }
}
```

In terms of postscript the corresponding routine is

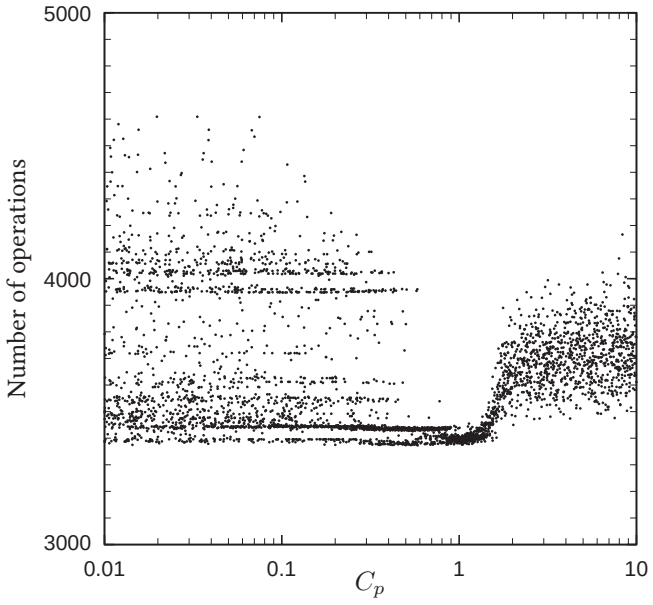
```
%   Incoming stack:
%   [array of x,y pairs] width scale
\special{! /polygon{
    gsw /points ed
    /ss points length 2 idiv 2 mul def
    ss 4 gt {
        newpath
        points 0 get points 1 get moveto
        0 2 ss 4 sub { /ii ed
```

```
            /x1 points ii 2 add get def
            /y1 points ii 3 add get def
            x1 y1 lineto
        } for
        closepath
        stroke
    } if
    grestore
} def }
```

It would have been not very complicated to have axohelp generate the postscript code as well, but much of the code existed already from version 1 and hence we left it that way. Hence the LaTeX binding:

```
%
% Draws a curve through the points in argument 1.
% The points are given as coordinates (x1,y1)(x2,y2)(x3,y3).....
% The curve is continous and continuous in its first and second
% derivatives. The method is linear interpolation of
%         quadratic curves.
%         Color name is argument 2.
%
\def\Polygon#1#2{%
  {%
    \SetColor{#2}%
    \ifcase\axo@pdfoutput
      \put(\axoxoff,\axoyoff){\AXOspecial{%
       [ \axoparray#1] \axowidth\space \axoscale\space polygon }}%
    \else
      \getaxohelp{Polygon}{"#1" \axowidth}%
      \put(\axoxoff,\axoyoff){\axo@pdfliteral{\contentspdf}}%
    \fi
  }%
  \ignorespaces
}
```

Axohelp can do something that TeX/LaTeX cannot do: read the complete .ax1 file and prepare the complete .ax2 file inside the memory and hence optimize it. TeX has only a limited capacity for such things. One can increase the upper-limit but even then there is a hard coded upper-upper-limit. In a first version of Axodraw2 it was programmed that the complete .ax2 file would be read at startup. This ran into trouble with the follwing type of plots:



There are 4000 points in this plot.

At the time we were preparing an article [4] that had about a dozen of these plots. Each point is drawn with a statement of the type

```
\Vertex( 68.08, 44.40){0.5}% 330
```

The system kept crashing and we were forced to translate the plots one by one and feed them in as .eps files. This is entirely against the Axodraw philosophy. In the end we decided to read the .ax2 file one line at a time. The loss in speed turned out to be rather small and it did solve the problem. It took however much more thinking.

About the examples

Back to the examples at the start.

The first example with the graphical formula is from a physics paper [5]. It is actually rather simple. If we take the code for the formula

```
\begin{eqnarray}
  \TAF(1,1,1,2,E,n,1)\ q\mydot q & = &
                (\tilde{N}+E-n-D+5)\TAF(1,1,1,1,E,n,1)
                +n\TAB(1,1,1,1,E,{n-1})
                \nonumber \\ &&
                +\TAB(1,1,1,2,E,n)
                +E\TAD(1,1,1,1,{E+1},n)
                -n\TAD(1,1,1,1,E,{n-1})
                -E\TAF(1,0,1,1,{E+1},n,1) \, ,
                \nonumber
\end{eqnarray}
```

there are a few macro's with parameters. One of these macro's is:

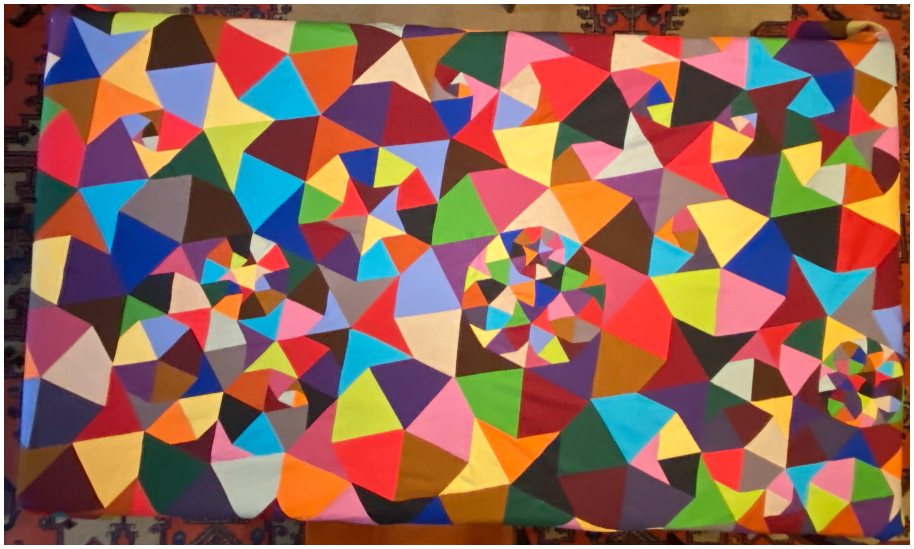
```
\def\TAB(#1,#2,#3,#4,#5,#6){
    \raisebox{-28.1pt}{
    \SetPFont{Helvetica}{14}
```

```
        \hspace{-22.5pt}
        \begin{axopicture}(90,59)(-15,-6)
        \SetScale{0.75}
        \SetColor{Blue}
        \CArc(40,35)(25,90,270) \CArc(60,35)(25,270,90)
       \Line(40,60)(60,60) \Line(40,10)(60,10) \Line(50,10)(50,60)
        \Line(0,35)(15,35) \Line(85,35)(100,35)
        \SetColor{Black}
        \PText(53,39)(0)[lb]{#5} \PText(53,36)(0)[lt]{#6}
        \PText(35,62)(0)[rb]{#1} \PText(65,62)(0)[lb]{#2}
        \PText(65,12)(0)[lt]{#3} \PText(35,12)(0)[rt]{#4}
        \SetColor{Red}
        \SetWidth{3}
                \Line(50,10)(50,60)
                \Vertex(50,60){1.3}
                \Line(40,60)(50,60)
                \CArc(40,35)(25,90,180)
        \SetWidth{0.5}
        \end{axopicture}
        \SetScale{1.0}
        \hspace{-7pt}
        }
    }
```

The Penrose file was generated by computer (of course). If not it can become quite difficult to make very big patterns. The computer generates the LaTeX file. Here is a little piece of it:

```
\begin{axopicture}{(560,400)(0,0)}
\SetWidth{1.000000}
\SetPFont{Courier}{182.03}
\SetOffset(360,160)
\Polygon{(-163.4721,-118.7694)(-139.6218,-126.5188)% \
         (-148.7318,-113.9800)(-148.7318,-98.4812)}{Black} % Dart
\Polygon{(-163.4721,-118.7694)(-187.3223,-126.5188)% \
         (-172.5821,-131.3082)(-163.4721,-143.8471)}{Black} % Dart
\Polygon{(-163.4721,-118.7694)(-163.4721,-143.8471)% \
         (-154.3621,-131.3082)(-139.6218,-126.5188)}{Black} % Dart
\Polygon{(-163.4721,-118.7694)(-148.7318,-98.4812)% \
         (-163.4721,-103.2706)(-178.2123,-98.4812)}{Black} % Dart
\Polygon{(-163.4721,-118.7694)(-178.2123,-98.4812)% \
         (-178.2123,-113.9800)(-187.3223,-126.5188)}{Black} % Dart
\Polygon{(-240.6531,-118.7694)(-216.8029,-126.5188)% \
         (-225.9128,-113.9800)(-225.9128,-98.4812)}{Black} % Dart
                .
                .
\Polygon{(288.3536,93.6918)(303.0939,73.4035)% \
         (312.2039,85.9424)(312.2039,101.4412)}{Black} % Kite
\Polygon{(38.5905,93.6918)(14.7403,101.4412)% \
         (14.7403,85.9424)(23.8503,73.4035)}{Black} % Kite
\Polygon{(38.5905,93.6918)(38.5905,118.7694)% \
         (23.8503,113.9800)(14.7403,101.4412)}{Black} % Kite
%\Polygon{(-328.2706,-205.1691)(328.2706,-205.1691)%\
%         (328.2706,205.1691)(-328.2706,205.1691)}{Black}
\Vertex(0,0){2}
\end{axopicture}
```

The generating C program takes about 350 lines and can make the pattern as big as you might want it. In this case it was a pattern for a table cloth of about 900 pieces of cotton in various colors. In the actual execution some local variations were made in the size of the pieces. The result became:



The example of the pattern for the t-shirt with long sleeves has an interesting problem. How do you make it such that the sleeve fits exactly with the body? Fashion designers use heuristic algorithms to draw lines that will fit more or less, but they can be off by a centimeter. There are special sewing techniques to correct for this. In our example the lines are drawn as the sum of a few Bezier curves and then the problem is that the sum of those (different) curves for the sleeves must be equal to the sum for the body. By making variations in the control points of the Bezier curves one can get the lines correct to small fractions of a millimeter. And the routines to calculate those lengths can be copied from axohelp, if the creation program is written in C at least.

All together it does not take much effort to make a program that for given sizes creates a perfect pattern in terms of a LaTeX program with Axodraw statements.

In the postscript version of Axodraw it is not very difficult to add commands. This can be done with an extra style file, e.g. sudoku.sty for showing sudokus together with their step-by-step solution with explanations.

It is a bit more complicated with the pdflatex version. The LaTeX part is not very complicated. The problem is when calculations are needed, because then the axohelp.c file needs to be extended. By itself that is not very difficult, but then the modified axohelp needs to be used and not the version that comes with the TeX distribution. This makes it a bit less elegant. To avoid this the current paper has been translated in the old fashioned way:

```
latex paper
dvips paper -o
ps2pdf paper.ps
```

This gave however the problem that dvips cannot handle .jpg files and the paper has two photo's included. Fortunately these could be converted with the GIMP program into .eps files.

The code for the sudoku is given by:

```
\begin{center}\begin{picture}(200,200)(0,0)
\SetWidth{0.5 }
\MakeBoard(10,10,20)
\ColorCell(2,6){LightBlue}
\ColorCell(2,8){LightRed}
\PutSudoColor(8,9,2){LightRed}
\PutSudoColor(7,1,2){LightRed}
\PutSudoColor(4,3,2){LightRed}
\PutSudoColor(1,2,2){LightRed}
\PutSudoColor(3,4,2){LightRed}
\PutSudoColor(9,6,2){LightRed}
\PutSudoColor(9,4,2){LightBlue}
\SetCoordinates
\PutCell(1,1,8)
\PutAllow(1,2,4)
\PutAllow(1,3,1)
\PutAllow(1,3,2)
\PutAllow(1,3,4)
\PutCell(1,4,6)
\PutCell(1,5,9)
\PutAllow(1,6,1)
\PutScratch(1,6,2)
\PutAllow(1,6,5)
\PutAllow(1,7,2)
\PutAllow(1,7,4)
\PutAllow(1,7,7)
\PutAllow(1,8,2)
     .
     .
\PutCell(9,5,4)
\PutAllow(9,6,1)
\PutCell(9,7,9)
\PutCell(9,8,3)
\PutCell(9,9,6)
\MakeBoard(10,10,20)
\end{picture}\end{center}
```

and it was completely computer generated. It is an example from a sudoku book that first explains many tricks and then gives 500 sudokus that were generated by a Monte Carlo technique. When one does this, most sudokus are rather trivial. Hence the program generated many millions of them and the more interesting ones were selected. There were also still quite a few that could not be solved with the techniques explained in the book, indicating that maybe there will be a sequel with very difficult sudokus. The book can be found at

`http://www.nikhef.nl/~t68/sudokus/book.pdf`

The origin of the name

In 1976, during my graduate study in Stony Brook, we did our computer work on the CDC computer in Brookhaven. We worked by means of a telephone line and the fast modem was 300 baud. I had a number of kinematics routines for my calculations and I was told things would go much faster if I would make them into a library. Hence I got the CDC manual. It had an example and the name of the library in the example was mylib. That did not look very creative. Hence, how to call it then? I looked around and at my feet was my dog. And the name of the dog was Axo. And there

was the name: axolib. That library has served many years and featured in many calculations. I still use it in lectures about particle kinematics. There are some rather unique routines in it. Years later it was a small step to call the drawing package Axodraw, even though at that moment the dog had been dead already for 7 years.

References

[1] J.A.M. Vermaseren, Comput. Phys. Commun. **83** (1994) 45–58.

[2] D. Binosi and L. Theussl, Comput. Phys. Commun. **161** (2004) 76–86.

[3] D. Binosi, J. Collins, C. Kaufhold, L. Theussl, Comput. Phys. Commun. **180** (2009) 1709–1715.

[4] J. Kuipers, A. Plaat, J.A.M. Vermaseren and J.H. van den Herik, Comput. Phys. Commun. **184** (2013) 2391–2395.

[5] Sven Olaf Moch and J.A.M.Vermaseren, Nucl. Phys. B573 (2000) 853-907.

J.A.M. Vermaseren
Science Park 105, 1098 XG Amsterdam, The Netherlands
t68 at nikhef dot nl
28-Aug-2020

Two Questions and Answer Sessions by Donald Knuth at FI MU

Abstract
In October 2019, the Faculty of Informatics, Masaryk University, hosted Donald Knuth as a guest who led two questions and answers sessions at this occasion, dedicated to the themes of Computer Science and art. Besides some background on these lectures, you can also find their transcripts in this article.

Keywords
Donald Knuth, Computer Programming as Art, Boundless Interests, Q&A

In October 2019, the Faculty of Informatics, Masaryk University in Brno, Czechia, celebrated the 25th anniversary of its foundation. Several events were held for this occasion, the most prestigious of which was perhaps *the Week with Turing Prize Laureates* (Sojka, 2019a) with Donald Knuth and Dana Stewart Scott as the guests of honor. At the end of the week, the Czech premiere of Knuth's musical opus, *the Fantasia Apocalyptica oratorio*, (Knuth, 2020b) took place, as discussed in the previous issue of the CS-TUG Bulletin. (Lupták, 2019)

This was not the first time Donald Knuth has ever visited Brno: In 1996, Donald Knuth gave lectures and has received an honorary doctoral degree at the Faculty. (Zlatuška, 1996) Twenty-five years later, Donald Knuth gave two lectures in the Q&A format inspired by the famous physicist, Richard Feynman. The titles of his lectures were *Computer Programming as Art*, and *Boundless Interests*. Both lectures were moderated, the first by prof. RNDr. Jozef Gruska, DrSc., and the second by prof. RNDr. Jiří Zlatuška, CSc. Questions to the speaker were also posed by the audience via the Slido web service.

In the first lecture, Donald Knuth confided to the audience about the importance of batch processing, the advantages of not reading e-mails, the deviousness of the „What is you favorite X?“ type of questions, Knuth's cooperation with the NSA, the renaissance of magnetic type data storage algorithms, and the relation between the sounds of a water fountain and a crying infant.

The second lecture was interleaved by passages from Fantasia Apocalyptica, played by Donald Knuth on a digital keyboard. In the lecture, you will learn about the musical side of Donald Knuth, the motifs used in Fantasia Apocalyptica, the reducibility of its composition to Constraint Logic Programming (CLP), how Donald Knuth grappled with burn-out during the composition, how he managed his time, the difference between programmers and *good programmers*, what Donald Knuth considers the worst program he ever witnessed, and also Donald Knuth's very own pipe organ.

Additional information, video recordings (Sojka, 2019b; Sojka, 2019c), and transcripts of the lectures (Szaniszlo, 2019a; Szaniszlo, 2019b) are available on the web of the Faculty. A video recording of the Czech premiere of Fantasia Apocalyptica is available on YouTube. (Knuth, 2020a)

Questions and Answers Session 1: Computer Programming as Art

Gruska: Good afternoon. Dear informaticians, we have today very special colloquium. We have the legend of informatics, father of analysis of algorithm, father of art of programming, father of many other books, father of TEX and so on and so on… I better stop, because I would spend probably all my time here talking about his outcome. He came here to answer your questions. Sessions will be so interesting, how interesting will be the questions you ask. There are two ways to ask questions. Either by microphone or you can use mobile. Ok?

I mentioned that this legend of informatics and one argument supporting that is when in 1989 there was the IT World Computer Congress in US in San Francisco, professor Knuth was invited to give the first talk, the most important talk. So welcome, professor Knuth. Welcome here, and you can start the session, colloquium, by asking the questions from your audience. Welcome in Brno. *applause*

Knuth: Prof. Gruska, do you have a question?

Gruska: Yes! 42 years ago I wrote you letter.

Knuth: Oh!

Gruska: I was waiting for answer. After one month I decided to call you. Secretary took the phone and said: "Oh, he's three months behind answering letters. However, he is next to me." So I could talk to you. Has that changed with your custom to respond to letters?

Knuth: Can you hear me? Testing… Testing, testing. Zero one, zero one. *laughter*

So the secretary I had 42 years ago is Phyllis Winkler who served me for many years, but she died about 15 years ago and actually retired before that. However, I go into campus at Stanford four days a week, and I have lunch with the students every Thursday and with the faculty every Tuesday and things like that… However, I found that one of the great secrets of Computer Science is called batch processing. So when I answer mail, I don't do it by interruption, but by batch processing. So I gave up email, I guess, on January 1, 1990, and I've been a happy man ever since. But the questions come in, and they're filtered, and I answer a lot of them all at once. So it turns out that that's necessary for me in order to work efficiently. But Phyllis was a wonderful protector. So she kept it possible for me to work efficiently for me all these years.

Gruska: And there's also another question, well, they ask, one student: Can you talk to Knuth? He said: "Yes, Friday night."

Knuth: Friday night? Ok.

Before I go on to more questions I want to mention that the whole inspiration for this kind of a session came about when I was at Caltech during these 1960s and Richard Feynman, our famous physics professor, would end every one of his classes the last day of class: Anybody can ask any question they wanted to. And so I started using that in my own classes, and I kept that up until I retired. And so now I find this as a best way to customise the lecture. So I will try to give an answer to basically any question, and I'll try to keep it short, so we can move to a variety of question.

On the other hand, tomorrow I'm giving another talk at 12:30, and it's also called Questions and Answers. Tomorrow I'm gonna have a piano keyboard, so that I can answer questions if they have anything to do with music because I think when you walked in here, you've got an ad for a concert that's gonna be given on Friday. So today let's not have questions about music, but anything else is fine.

Then I also brought this with me. This is a prop for a paperback book that I published, I don't know, 2–3 years ago, which is the middle third of Volume 4B of The Art of Computer Programming. So far I've finished Volume 1, 2, 3 and 4A and I'm working on Volume 4B. I started writing it in the middle, and this came out. It's called satisfiability, and the big six on it here says Volume 6, Fascicle 6. Fascicles 0, 1, 2, 3 and 4 were part of Volume 4A.

Now the reason I'm saying this is that right now, maybe as we speak, Fascicle 5 is being printed and it will be available in bookstores a month from now, and you will love it. It's a wonderful book. Ask your parents to get it for you for Christmas. *laughter* And the main thing is that it's, I would say, maybe eighty percent of it is about puzzles or topics that are often considered not only worthwhile but fun. I sort of have been waiting all my life to present this material in showing how puzzles are very relevant to learning how to be a good computer programmer. And so that's Fascicle 5 gonna be coming out in a month. So that's end of advertisement.

Gruska: There is a question: What is your favorite unsolved problem in Computer Science?

Knuth: My favorite unsolved problem in Computer Science… Ok, well… I guess, personally, what is the worst kind of question to ask? And I'm sorry, but it's a question that starts by "what is your favorite X?". Because it's really hard to say what is my favorite X: almost always "what's my favorite algorithm?", "what's my favorite… relative? – Do I like my son better than my daughter?", and so on… So I hope not too many of the questions today are gonna be "what's my favourite X?", however, I don't wanna duck this one.

So unsolved problems in Computer Science. It depends on who and whether I think it's gonna be solved or not. So in order to be a really favorite problem, it would one where I expect that when I tell you what it is, then next week someone in the room will solve it. So often my favorite problem is one that I've just thought up. And last week I thought up a problem that I mentioned to one of Dan Král's students on Sunday night, and so I'm not gonna repeat that one now, so he can work on it and solve it for me.

Of course, the most famous unsolved problem in Computer Science in a sense that it's got a million-dollar price associated with it is a question "does P equal NP or not?" Someone is gonna ask me about that anyway, so I might as well spend one or two minutes on that. So the question is: The P is the set of all problems for which we can solve in polynomial time, and NP is a set of all problems for which we can verify a solution in polynomial time, more or less. And so the big question people say is "Does P equal NP or not?" Now, I wanna amplify that question. So let's try to be specific. It's known that the question "P equals NP?" is equivalent to saying "is there a polynomial-time algorithm to solve one particular problem?" Let me take satisfiability, for example because I held that up.

So there's a problem called 3-SAT which says: I have n clauses and each clause is of the form $(a \vee b \vee c) \wedge (d \vee e \vee f)$ and so on. Various clauses like these and all together [logical] and of these. And a, b, c, d, e and f are actually $x \vee \bar{y} \vee z$ or something like that… a, b and all of these things are Boolean variables that are either negated or not. And the question is: Can we satisfy all these clauses simultaneously? Is there a way to say that x is true, saying y is true and z is false, and all of the clauses are gonna be true? And that it's satisfiable if and only if there is a way to set these variables.

So when you look at this problem, you say: "Oh, sure, I can easily solve this problem." However, I don't wanna say if you've solved it tell me how you did it because a lot of people have told me that, but it was a waste of time. A lot of people think they've solved this problem, but they didn't.

The question "P equals NP?" says "is there an algorithm that solves 3-SAT to some constant steps?" I wanna ask another question, and that is: Could we know an algorithm that solves 3-SAT? Some people think that these are the same questions. If an algorithm exists, certainly, we would know what it is, right? However, that is a giant step to go from saying that there is an algorithm, and there is an algorithm we actually can find. A lot of things are proved non-constructively, so it can go several ways. One is that we have "yes, there is an algorithm, yes, we know it". Or it might be "yes, there is an algorithm, but no, it actually is beyond our grasp". You can never

write it down. It exists up there, but you need to be God in order to know, actually to use it. And then there is a case "well, there is no algorithm". Well, then this [the fourth cell in a schematic combinatorial table] had better be no. I'm not gonna have an N and Y in here.

Gruska: Would you be happy to have non-constructive solution?

Knuth: Well, that's what I believe is probably true. I'm not sure how happy I would be, but it could be that the total number of algorithms is huge, and if it doesn't exist, it's a completely different question. It's quite likely, but hardly anyone thinks about this puzzle. It's quite likely that the algorithm exists but only because there's only finitely many reasons why it doesn't exists. And that's not gonna tell us much. There are many cases of non-constructive things where we know, for example, that there's a winning strategy in a game of hex, but nobody has any way to actually win hex. It's just known that there is a winning strategy for the first …

Gruska: So next question…

Knuth: Yeah… *laughter*

Gruska: Do you still write any code? If so, why and which programming language?

Knuth: In this case, I could even say what my favourite programming language is. *laughter* When I wrote Fascicle 5 I probably wrote about 600, I don't know, many hundreds of programs and every week I write on average at least five. Most of programs are rather short, of course, but some of them get to be fairly good size. The language I use all the time is called CWEB. It just really works for me. It comes installed with Linux, and so I've got many dozens of CWEB programs as examples on my website. If anybody wants more information about any of the ones that I didn't put on the website yet, I'm glad to put it online.

It's a combination of TEX and C. It's so much better than any other way to write programs, but I'd better not get started on it. To me, it's the greatest outcome of all of my work on TEX. The fact that I now have CWEB in order to solve lots of problems.

Gruska: Ok, a related question. Did you try to write program for quantum computer?

Knuth: Ahaaa! Quantum computing is something that I have absolutely no intuition for. If quantum computing turns out to be the best way to go and everybody switches over to quantum computing, in a way, that will be the best thing for me, because then I'll have more time to write my books. Because I'm never gonna write anything about quantum computing. I only promise to write about things that were known in 1962 when I started a project [The Art of Computer Programming]…

Gruska: Another question is such that you may have a difficulty to answer. The question is: What were the research problems you worked on during your cooperation with NSA? *laughter*

Knuth: So I spent a year before going to Stanford working on code breaking, and I'm not allowed to tell my wife what I did during that year. *laughter*

Gruska: She's not here! *LAUGHTER*

Knuth: But this is being recorded, and maybe she would watch the recording.

Gruska: [Disappointedly] Oh…

My question is: What was your subject in your thesis? PhD. thesis. And did your advisor help you?

Knuth: I was lucky to work with Marshall Hall at Caltech, and so my thesis was about a general area that now would be called combinatorial designs. More specifically, finite projective plane. This is an arrangement of points and lines that are abstract. There is a parameter n, and I think it was… $n^2 + n + 1$ point, and every line contains $n + 1$ points and every point is on $n + 1$ lines and any two lines intersect at exactly one point. So this is like a projective geometry, but finite projective geometry, instead of the projective geometry of the sphere or something. So my thesis was to show that

certain kinds of finite projective planes do exist and they hadn't been known before to exist. The whole area of designs is to find families of sets that have interesting properties that might be useful in application.

Now the interesting thing is that over the year since I graduated, I applied almost every branch of mathematics that I've ever heard of to computer programming except the theory of design. Because subsequently, we've found out that we can do better with random choices in almost all cases instead of trying to find these very rare patterns that sometimes exist. It was very good training, and my advisor helped me in the following way: I was working on another problem and one morning as I rode up in the elevator, got to my office and said hey, I betcha I can solve this particular problem I just heard about. And my advisor said: Ok, that's your thesis.

Gruska: Another question: Vim or Emacs? [The question wasn't answered at this moment. But see a later mention near the text "Tabs or spaces" in this transcript.]

Knuth: What do I think about Artificial Intelligence? Ok. Do you think it could be dangerous?

Certainly, I prefer real intelligence to Artificial Intelligence. I have always felt that the working ??? AI has been at the cutting edge of Computer Science. Of the last sixty years, the people working on the challenging problems of Artificial Intelligence have come up with many of the most important innovations in the field. I always regarded it as something that tells us how to stretch what we know and invent better algorithms rather than as something where I would actually use the algorithms afterwards and believe that.

In other words, suppose we develop a really good algorithm that decides whether or not a student ought to graduate. Should I let the computer decide which students graduate or should I try to understand their working and see if it has some value?

Right now, the question about the danger is extremely pressing and especially with respect to military applications. There is this really frightening movie, short video… I can't remember the name of it. Came out about three years ago. Where you could pretty much, with today's technology, you could program drones to kill anybody you wanted to. The scenario I should remember, some of you will maybe remember the title, but my friend Stu Russell at Berkeley was one of the people behind that film. Essentially it's already possible to do these horrible things, so we gotta find some way to keep that from happening. Stu has the best idea so far about how to prepare for such dangers, but still, I'm afraid his ideas are not satisfying me because they depend on assumption that human beings are rational. And the more I read about these days, the less and less I believe that human beings are rational.

It's very serious to see how to restrain the things that we don't understand and figure out how, because with the new techniques we are able to solve many problems, wonderful problems in all branches of science that we weren't able to solve before. The ones that are encouraging to me are the ones where there's no enemy involved. Like somebody trying to defeat us in the experiment, but the only enemy is that we're battling ignorance, we're trying to find some pattern in the way stars work or something like this, the way biology works, how to identify diseases of different kinds… And these machine learning techniques are wonderful. Even though we don't understand anything about how they work. But you use the same algorithm for something where there's adversarial interest involved, then everything gets bad.

Gruska: Another question was very nice, and I think you will like it: Could you tell us the story of TEX from the very beginning to implementation?

Knuth: Ah, yes. *laughter* Yes. In short, I found out that computer technology had changed so that the work that was once done by hand with hot metal was no longer being done. It was replaced by new technology which was based on photography. And the new technology looked awful. I saw the proofs for the new edition of Volume 2 of The Art of Computer Programming, and I was almost sick. I didn't wanna have a book that looked like that. And then a couple of weeks later I learned that computers

might be the answer, because somebody working in Southern California had found out that actually with digital methods, with pixel zeros and ones you could potentially make books that would look just as good as the real printed one.

So I got in an airplane and flew down to South California, talked to the people there and decided to change my sabbatical plans for the next year where I was gonna study combinatorial algorithms. Instead, I decided that oh, I can now solve the problem with the books if I only write the computer program that makes patterns of zeros and ones that say what should be on every page of the book. Well it took a little longer than a year, but that was the beginning of my work on typography.

Zlatuška: So the proofs for Volume 2. That means Volume 1 was already printed not using TEX? Does it exist?

Knuth: No. Volume 1, 2nd edition have already come out.

Zlatuška: The first edition you ??? about.

Knuth: Volume 1 would have come out looking bad at some point, but I was revising Volume 2, and so those proofs came in. What happened, is the technology went away from hot metal, basically monotype setting, and actually in Eastern Europe was the only place left for people who were still doing monotype during the 70s. So I have a translation of my book in Hungarian that as done by these hand methods still look good in the 70s, but the books that were printed… if you look at all the journals of mathematics that were printed in the 70s you see what I mean. So I was feeling bad about until I realised it was just a matter of programming. The machines were there that would make the book, but I need to get the pixels figured out. So I spent a long time in the library looking at everything that I could see about how to make good-looking books, and I brought the experts to Stanford, and we worked together on it for a while.

Gruska: How many months you worked on TEX?

Knuth: It's hard to say because I was also doing a few other things, but at one point I took a leave of absence from Stanford for a year because I found that working on software was harder than writing books. You can write books, you can write papers, but software involves much more of your brain, and I couldn't swap in and swap out so much without taking a year off of teaching and getting TEX done. Then I could go back and resume the other schedule. But in calendar time I finished this five-volume set of books called Computers in Typography. I finished that in 1984, I have started the project in 1977. So that's seven years. Then I came back to it in 1989. I came back to it because I didn't realise that people were gonna be using it for typesetting strange languages like Czech. *laughter* You know, you have accents on letters. Wow. So I went from 7-bit to 8-bits in 1989. Took a year.

Gruska: As far as I remember, when working on TEX every evening or every morning you wrote down what was good, what was bad. Did you make use of these comments?

Knuth: Well, I got a paper out of it. I have a paper called "The Errors of TEX" which I think is a good idea for everybody – to keep track of what mistakes you make. Programming is too complicated, you can't get it right all. And I wanted to find out what were the kind of errors that I made and also learn something about patterns and that I could change my actions. I kept this log showing every century(???), every major non-trivial change that I made, some of the trivial ones too, to TEX and to METAFONT over the years. Then I was able to get a good understanding of the scale, how important are certain kind of errors. For example, people say: goto statements are bad. Look at my history with TEX – sure enough, I made some errors, because I used goto statements improperly. However, I had also used every other kind of statement improperly too. Every statement – assignment statements are bad, conditional statements are bad, everything from that aspect can be misused. But I did learn something about which kinds of things to avoid and the corrections were not only to fix errors but also to improve user interface and things like that.

Gruska: Next two questions are pretty ??? so please ???.

Knuth: All right so… What would you advise to your 25 years [old your]self?

So 25 year, that would be… 1963. That's the year I got my PhD. … I decided that year to become a college professor. I actually been offered the year before when I was 24 to drop out of grad school. Essentially I was offered a salary of $100,000 a year plus an assistant. Now in 1962 that would be like $10,000,000. It's not anywhere near Bill Gates's salary, but anyway I knew that my role in life, my interest was going to be to work with students rather than to maximize the amount of income that I had. On the other hand, I'm not saying everybody should make that decision. Anyway, at 25, that's when you want to start becoming stable and making long-term decisions that you're gonna live with.

Knuth: Next question: Anonymous:

Gruska: How being a Christian affected you as computer scientist?

Knuth: It's hard for me to say what it would have been like if I had been born into a different family, but certainly, I guess, one of the ways, my Christian upbringing with respect to work on crypto and questions of computer security. I'm not very good at doing work on cryptography, because I'm not as sneaky as the people who are trying to defeat these things, but with respect to security all my life I had this idea… I'm sorry, not security, privacy… I always had the idea that everything that I do is known to God, so I don't have absolute privacy. And I didn't understand until later that there are people who think that nobody should know what their thoughts are. So it makes it harder for me to understand, but some people concern about privacy, although of course, I don't want my thoughts to be used by the devil, by something that's going to exploit me, but I'm not very comfortable if there was something beneficent watching over what I'm doing.

All these things, I guess I should say, I'm very happy that there are parts of my life in which there's no mystery, and I can prove that something is right, but I wouldn't be happy if there was no mystery whatsoever, so I appreciate the fact that there are things that I'll never understand, so it teaches me some humility that I shouldn't expect to understand everything. So I don't claim to understand everything, and I'm very happy that God did not make it possible to prove or disprove the existence of God.

Gruska: Related to this question: Let us assume that you will live still 30 years.

Knuth: Do I? What now? … Oh goodness. *laughter*

Gruska: By famous visionary Kurzweil at that time we should have laptop with information processing power better than all human brains. What would you do with such laptop?

Knuth: I would certainly try to finish The Art of Computer Programming. Let me rephrase your question. How do I want to continue, you know, what should I do the next week and the week after that? How do I want to continue to live? I have to be watching when am I gonna start going senile. At the moment I don't think I've reached that point yet, but it might come to the point where I should stop writing The Art of Computer Programming because I'm starting to write stupid stuff.

Gruska: In which area of Computer Science or mathematics do you see the most potential?

Knuth: In which area… This is one of these favourite questions again. The much harder question would be: "Which area does not have much potential?" because I think everywhere I look, I see potential. The problem is really have to go to sleep at night not using the knowledge we have. Everywhere I look, I see that it's not saturated yet.

Gruska: Additional question is pretty philosophic: What your opinion on Curry-Lambek correspondence? Do you think mathematics is constructed or exists independently?

Knuth: I guess I'm a Platonist in the sense that I'm discovering things that are there already. The stuff that's there is all consistent with my own attitude that these truths are there and I just am learning a few of them at a time. There might be an algorithm that solves 3-SAT, and that would be there. In fact, I'm not sure I even understand how I could be a non-Platonist.

Gruska: What was the most valuable ??? you learned during your career?

Knuth: Not to use email? *laughter* There we go.

Gruska: Is too late for 20 years old students to start learning real mathematics and programming?

Knuth: A 20-year old student? My goodness, no. I didn't now that much math when I was 20. It depends on what you've seen so far and the teachers you've had. But I consider life to be a binary search where you find out things that are relatively easy for you, because of the unique experiences that you've had, and you try some things, and some things work, and some things don't work. Then you keep learning more about yourself. I'm 81 years old now, I'm still not sure exactly where to go, but I keep trying different thing. I have given up on quantum computing, though. *laughter*. I tried to understand quantum computing, and I know people who do understand quantum computing, but I ??? it's not me.

Gruska: Do you believe in non-locality in physics?

Knuth: I understand a few things about multiverse and things like that. For example, there's no way to distinguish between whether or not this lecture I'm giving now is simultaneously forking into many different things and so each different incarnation of it will have a different series of questions, and I'll give different answers to different questions and so on. And all of this idea that there are these all the different universes all simultaneously existing is consistent with quantum mechanics.

Gruska: Biggest challenge of becoming a good programmer?

Knuth: What does it say? Spaces… Tabs or spaces? *hehehe laughter*

So I use Emacs for my hacking, and I always untabify ??? but I also read a lot of programs that other people have written, and I start out with changing all the tabs to spaces. Of course, I'm using CWEB, not Python.

Knuth: Next… Some of the exercises are known to be open research problems. Yes indeed. If the number is 46 or higher, it's something where, as far as I know, it hasn't been solved yet.

Knuth: Has anyone ever contacted you that they have solved one of them while reading the book? Yeah. People look at them ??? I do want to point out that a lot of people haven't been looking at those lately, so … I'm sorry…

Gruska: Excuse me… We make break for 5 minutes because they need to change [the microphone].

Knuth: So we had a celebration at Berkeley, I guess a month ago, celebrating the life of Dick Karp and at that time there were a dozen speakers, and I decided I would say something about the Karp's work that the others weren't going to talk about. So I looked again at Volume 3 of The Art of Computer Programming, where Dick had told me about some things he never published that applied to sorting on tapes. Nowadays people don't use tapes for sorting, but when Volume 3 came out, this was one of the biggest topics in all of Computer Science. People were saying one third of all computer time was spent on sorting, and people had these tapes, and these were must have to have in the book. But since we don't do sorting that way anymore, I should rip out all this, I don't know, sixty pages or something of Volume 3. And then people say "no, no, don't take it away, because we're just finding out there's a new memory kind of being invented which is rather similar to tape and so these old techniques are gonna be good!"

Anyway one of the things about magnetic tape is that you can read it forward and backwards, so you can write information on the tape, and then you can read it in the other direction, and that would be much faster than rewinding and reading back forward. And Dick Karp worked out a beautiful theory about patterns of using tapes for sorting that he showed me at lunch one day and I put it in my book, and he never published it. So I showed it to the people at Berkeley, and as I was looking through this I came across a research problem, you know the level 46, which it seems to me is ripe for solution now. 50 years have gone by, and people know a lot more math than then, so I suspect that there are dozen problems in there that are just waiting to be solved. So you can go through, you have to spend a little time paging through and checking out the numbers. And if it's 46 or more then think about it and say "hm, I wonder if I can solve this now", because people haven't been doing that systematically.

Zlatuška: How many girls you seduced because of Computer Science?

Knuth: How many girls have I had thanks to Computer Science?

Gruska: Forget it. Forget it.

Knuth: So, in fact, I had 28 grad students, but none of them were female, *laughter* but I did serve on many committees and many others and so on…

Gruska: Here is better question: What's the biggest motivation that kept you going through career?

Knuth: I guess it's the example of my parents which was always to somehow be a servant, to see how I could be of use to somebody else. My father's name was Erwin. He had an informal, I guess you'd call it a startup ???, he is a one-person operation, and he would do services for all churches and schools and a few other nonprofit things like this, and he called it ERW service. He thought it was clever he could write the word service but make ERW large, so would say ERW service. But anyway, that epitomized all the philosophy that I grew up with – to be a service. If I got to a point where I thought that I couldn't be of use to anybody anymore, I told my children not to keep me alive just to make me happy, but if I get to a point where I don't recognize them or anything like this… How do I express this… There's a novel, came out less than 20 years ago by P. D. James and it was about… the world… the people discover that from now on all women in the whole world would be infertile and so there will be no more children ever born again. So humanity was eventually going to die out. So what did people do? That was very devastating to me, to imagine what it would be like if I was in a situation where I could not do something that would be of use to people, more than my immediate family, but to people in the future.

Well there's a short story by… Oh, goodness… Okay, who's the greatest Argentinan author?

Zlatuška: Borges.

Knuth: Borges, yes, of course, Borges. So he has this short story. I think it even takes place maybe in Czechoslovakia, I don't know. But anyway, it's a story of a person who is a playwright, but he's facing a firing squad, because of his political views. And just as the guns are aimed at him and so on, he prays to God, and he says: "God, I have to finish this play I'm working on. Please, make time stand still so that I can work out all the details and figure out exactly what should happen in this play." So God says okay and time stands still, and this playwright solves the problem, he figures out exactly what's the perfect play. And then he's shot. So nobody ever gets to see what happened in the play. It was just that the playwright himself was able to solve that particular problem. So that's the opposite of my own way to do it. It has to be something that I do that somebody can use. I hope that's making myself a little clearer what this idea of service is.

Gruska: There are quite a few people that try to put the idea of formal method into programming, software development… What do you think about that?

Knuth: When you're putting ideas into formal methods, it forces you to understand what the ideas are. You don't realize what you don't know until you try to formalize it some way. So it's a great educational experience. On the other hand, I guess I said a similar thing about Artificial Intelligence a while ago that I consider it as a very useful way, a source of good problems and continuing to learn. But then I was less interested in actually using the programs afterwards because I knew that they would only be approximately right.

I once had a language called SOL, Simulation Oriented Language, and the idea was that it would be it would be pretty easy to write models for discrete systems. And you could simulate the system. As I was working on this language, I looked at a lot of different applications, and in each case, I found out that the idea of formalizing the application and putting it into this language was a wonderful educational experience. I can build models of things to simulate, but I learned much more writing the model than I did actually running the model afterwards. So I almost thought I should take output statements out of the language. The people only use language for formalizing their model instead of for actually running it and believing the answers that they get afterwards.

Nobody seems to understand what I'm saying, but anyway, I believe the main advantage of formalism is educational rather than actually having a payoff afterwards.

Gruska: Next question: For how long can you program or read papers per day until you are exhausted? How do you relax? Do you relax sometimes? *laughter*

Knuth: Yes, but in fact, I'm relaxing right now. *laughter* At night, I have to take my mind off whatever I'm doing. So I read James Bond or something not really heavy literature. I'm reading right now a novel by Frederick Forsyth called The Odessa File which is story about the German SS in Latvia… what you'd call a thriller. Some kind of story like that and I go right to sleep. I read novels at the same speed as I read a math paper. *laughter* I don't know any other way to do it, so I don't get through that many books. However, when I do come across a book, that I think is especially nice, I put it on my website. So I think if you look on my website and under… I think, there's a Frequently Asked Questions and it says "so you're retired" or something like this. You click on that, and it'll tell you, I think, maybe three dozen books that I think were special to me.

Gruska: Did you work hard when you were student?

Knuth: I was a machine. I was a problem-solving machine. Somebody said: Okay, Don, do this, and I would do it. And I didn't start reading for pleasure probably until I was about thirty years old. I was given an assignment, and I was a good boy. So I did it.

At least that's the way I remember how it was. I don't know how it really was. I think I also was a… they might call me a wise guy. I mean, I was cracking jokes and not paying too much attention to what I was supposed to do.

Gruska: Did you watch westerns or detective stories?

Knuth: What about detective stories?

Gruska: Did you watch westerns movie or detective stories?

Knuth: I certainly watch a lot of movies, but, by the way, I might as well mention one thing. I guess it was a month ago when I saw again a movie. It's called Double Indemnity. It came out in 1944 or something like that. It's one of the earliest noir movies, and it stars Fred MacMurray and Edward G. Robinson and Barbara Stanwyck. That movie has special significance for me because when I was writing The Art of Computer Programming, that movie was playing almost every night on The Late Show. As I was typing The Art of Computer Programming, I would have the TV on, and I would see Double Indemnity over and over again. So The Art of Computer Programming was written largely to this movie Double Indemnity. The music to Double Indemnity was written by Miklós Rózsa, and it's haunting music that I hadn't

remembered how haunting it was until about ten years ago. I saw Double Indemnity again at a theater and immediately when they showed the title, and I heard this music, and I said: "Oh my god, what powerful music is it." So I've included music from Double Indemnity in my piece that's gonna be played on Friday. Although I'm not sure if I'm gonna be sued for this. But one of the things that occurs, you can check it out by watching the movie.

Gruska: Didn't you have idea to write science fiction?

Knuth: I talked about writing it or? I enjoy different kinds of science fiction, of course, but I haven't had time to… I wrote this little book called Surreal Numbers. I kind of think of it as a little bit like opera in the sense that opera is good music to a little bit of a plot. My book Surreal Numbers is good mathematics to a little bit of a plot. The characters in this story work on a math problem together, and it makes a little plot, but mainly there's beautiful mathematics that they're discussing.

If I live long enough and finish The Art of Computer Programming, I'd like to write some science fiction. One book I'd like to write is where the story is told by ant colony. Not by an ant, but by ant colony. There are tens of thousands of ants, and somehow they cooperate with each other and so that they form a consciousness and so I think there ought to be a short novel that's told by an ant colony.

The other thing is a short story something like… you could call it The Fly. Now, Mark Twain in one of his books, I think it's called The Mysterious Stranger, there's a character in there representing the devil, and early on in the book the devil opens a window, and a fly goes out the window. And he said: "Because I let this fly out the window, there's gonna be war next year." He's predicting what they call the butterfly effect of chaos theory. All kind of things are codependent.

So maybe people have seen this movie Run Lola Run that came out of Germany a few years ago. It tells a story, three or four times the story starts out exactly the same way, but then there are three or four different, completely different endings, just because of little changes in time. So I think it would be fun to write stories that… Well, it's not one story, but it's two or three stories that all start out the same, but end completely differently. But on the other hand, did this movie, its German name is Lola rennt. Run Lola Run in English. So the author of that movie already did it, what I was planning some time to do.

Gruska: So, please, ask questions. Yes, will be finishing. Maybe those who don't have mobile with, they should have the chance also to ask questions.

Knuth: Yes!

Student: You have a trouble ??? distractions or maintaining focus when you're working on a problem during the day? And if you do, how do you sit down and focus and do some piece of work during the day? You have like a ???.

Knuth: If I understand, you're saying how can I focus on one thing instead of many others? I don't have all the distractions of modern day. I don't have the radio play, music blaring ???. I found that, for example, if my job for the day is to do something like proofreading, then it helps me to have some kind of music like Telemann or something playing in the background. It sharpens my mind. But if I have Bach playing the background, it's too much, and everything goes away. So part of it is having no distractions, no interruption.

When my children were babies, they would be crying. I had this problem: How am I gonna concentrate on writing a book when the baby is screaming? The answer was white noise. I had a waterfall, and I could turn the fountain on and crying plus white noise equal white noise. *laughter*

But in general to concentrate on one thing I collect a lot of material, all on the same subject. I'll read thirty papers about that subject all at once, instead of reading about many different things. So I spent a lot of my time filing things away to be read later. Batch processing is very important.

Sochor: My question is: Do you prefer screen and keyboard or paper and pencil?

Knuth: I love that question because it turns out that I learned when I was in college that I could write a letter home to my parents faster with pencil and paper than on a typewriter. The reason was, actually, because I'm a good typist, I type faster than I think. *laughter* I had actually gone to typing school, so I can type so fast that it's a synchronization problem. *laughter* I'm not ready for it. But pencil and paper for me is a perfect sync with my thought process. So I make the first draft of everything in pencil paper, but then I go to the computer, and I polish it, and I edit for style. And I'm typing on it ??? I do that.

Gruska: So, you don't have to think, you just looked on your fingers.

Knuth: That doesn't work for me.

Student: So my question is: How has your stay in Czechia been so far?

Knuth: In fact, I wanted to say at the very beginning that it's a thrill for me to be invited here to celebrate your 25th anniversary. It couldn't have been better in every way, and I'm really enjoying this week. I was babbling over with enthusiasm for that, but I forgot about it at the beginning.

Of course, I've only had five/six days so far, but each one has been a joy.

Gruska: Ok, I think it's time to make break. The break will be quite long. Continuation will be tomorrow. Thank you, professor Knuth.

applause

Sojka: Come tomorrow and those who want to make a photo with Don and the faculty, the photography …

Questions and Answers Session 2: Boundless Interests

organ music playing

Zlatuška: Ok, ladies and gentlemen, dear colleagues, I would like to welcome you at the second session of Questions and Answers with professor Donald Knuth. Today, again, there is a possibility to put questions over your phones or whatever connection this works (???). And today's topics are not limited to just Computer Science, but also to music.

We invited Don for 25th anniversary of Faculty of Informatics. Not only as the first honorary doctor of informatics of this university and this faculty, but also as personality who transcends boundaries, and especially we felt that it would be interesting to have him here also as a musician.

I believe he's got at home not only his work office devoted to The Art of Computer Programming and informatics, but also to music. He built his own organ in his house at Stanford, so this other Donald Knuth, I believe, will be for many people as interesting as Donald Knuth, the author of The Art of Computer Programming. If you look at that from other perspective, I feel that the idea of programming as an art—Don quoted that yesterday—is something which resonates with this second personality, but I hope that there will be also questions concerning this part.

Now. I thought Donald travelled with this, but he travelled so light, but it would be impossible so… The floor is yours with the questions. Any questions from the floor?

Knuth: Right, so, hello, everybody. I want to thank again for the wonderful hospitality this week, and before I forget, I also want to mention that there's another lecture today by Dana Scott. I think 4:30 in the afternoon. So, you know, if I give a bad lecture, you at least get one good one today. *Zlatuška's laughter*

The other thing I wanted… Before we start with new questions, I had a couple things to say. First of all, I don't know if you've ever given a lecture, but the night after it you wake up in the middle of night saying "Oh, I should have said that!" and so I thought of at least two things that I wish I had said yesterday.

So in the first place, I was trying to explain why the question about P versus NP is not as simple as people usually think when they talk about whether there exists an algorithm that runs in polynomial time, and they think it's the same as saying that we could know an algorithm that runs in polynomial time. But there's a big jump between that, and then someone asked me later on about the Artificial Intelligence. So it occurred to me during the middle of the night that one way to think about it is this:

I want to give an example where maybe there exists an algorithm to decide the 3-SAT problem in polynomial time. But we won't know what the algorithm is. So imagine that we have a problem of size n, and we build, let's say, $(n^{1,000,000})$-state, some kind of a neural network that has $n^{1,000,000}$ neurons. So this is a polynomial number of things. Just add a few more [zeroes to the exponent]. So now we trained this neural network on lots and lots of SAT problems. You know, we've got great advances now in machine learning theory. Now I feed it a new SAT problem, and how do I know that it's not gonna solve all of them? In fact, in order to prove that, we couldn't possibly train this neural network. In order to show that P is unequal NP, we would have to show that there's no way to train this neural network to do it.

And that's a situation that we're faced with. Right now, when we do train neural networks, we've got something that solves the problem, but we have no idea what the network is doing inside. And that might be a way to think about the difference between an algorithm existing and an algorithm that we know what it is. Ok. Do you have a comment on that?

Zlatuška: If I may.

Knuth: Yeah.

Zlatuška: Could you extend this problem to the problem of human mind? The human mind and limits of human mind?

Knuth: Okay.

Zlatuška: Because that *laughter* obviously… That's obviously similar problem.

Knuth: Yeah.

Zlatuška: So what's your idea about limits, the capacity of human mind? You know, the ideas like mysterianism, and the existence of problems which are inaccessible to human mind?

Knuth: In that case, we actually have rigorous proofs, because there are incompressibility theorems. So we know that there are things that cannot be made small, and so the human mind can't possibly understand something that has more states in it then there are, well, let's say, protons in the universe or something like this.

Zlatuška: So, something between man and the God.

Knuth: Yeah. But in one of my papers, I have a number that I called… I don't know, I forget… I shall call it Super K, which is also the name of a breakfast cereal in America, but anyway, I needed a special font in order to do it. Like I wanted to print it only in color, but if you look with a magnifying glass at the paper where I talk about Super K…

Anyway, this was a number that was, I forget, it was something like $10 \uparrow\uparrow\uparrow\uparrow 3$, which is defined in terms of the [Knuth] arrow notation. Anyway, I was once giving a talk at Livermore Lab where they were trying to impress me by how big their equipment is, so I said: "Well, here, let me show you some big numbers that are bigger than you ever thought of before." When you try to think—what is this number Super K—this simply means *writing on blackboard* that you take ten… let's see… quadruple arrow I don't even remember the thing… Anyway, $10 \uparrow\uparrow 10$ is equal to $10 \uparrow 10 \uparrow 10 \uparrow \ldots$ and then if you want to go to triple arrow, then you simply do this that many times. Pretty soon, it gets mind-boggling.

But still, this number is finite, and almost all numbers are bigger than this. *laughter* You get a little idea that even though computer scientists stick to studying finite numbers, we aren't limiting ourselves too much. There's still a lot of interesting stuff down in… I don't necessarily insist on immortality, I would settle for living this long. *laughter*

Zlatuška: And if I may misuse the moderator's role, I would like to ask about music a bit. Within Fantasia Apocalyptica, you have lots of quotations from… or sort of inspirations from other authors. And what seems to me really fantastic is the scope. You go from Gregorian chants to authors whose music is sort of dramatic, like Olivier Messiaen. And incidentally to see quotations from Olivier Messiaen and Bruce Springsteen at the same time, well, both of them are favorites of mine, but what seems to me interesting, just to combine those absolutely different styles which are usually considered incompatible. How did you solve this compatibility problem?

Knuth: So come on Friday, but what's your opinion of rap music? *Zlatuška laughs* Because there's a quotation from Eminem as well. *laughter* And of course Dvořák. There are a few notes that are actually original with me as well. I wanted to mention that now before I forget because it ties in with Brno. When I came 24 years ago, my first visit to this part of the world, as my plane was approaching the airport in Prague, I woke up that morning with a melody in my head. And I wrote it down. Probably in the airline magazine or something, but anyway, I wrote it down, and I kept that piece of paper, and I saved it. I sort of thought of it as a Prague theme. Because really, I woke up and here was this melody and I wanted to write it down before I forgot it.

So fast forward twenty more years, and I'm writing Fantasia Apocalyptica, and I came to a point of the piece where I'm trying to represent Chapter 21… I guess I gotta go back. The point of Apocalyptica refers to Apocalypse, the last book of the Bible. I'm trying something new in this piece, which I don't think had been done before. To make a fairly literal translation of an original Greek text and convert it into musical equivalent. So I find more than hundred fifty motifs for things that are repeatedly used in the Greek text. The same sequence of motifs occurs in my piece…

Zlatuška: You consider yourself Wagnerian?

Knuth: Yes, Wagner had motifs, but he never told anybody what they were. So the differences is…

Zlatuška: He wasn't university teacher, so…

Knuth: For example, the motif for war is staccato. The motif for angels is an arpeggio. *keyboard playing* The motif for God is a three-note melody. *keyboard playing* If you're referring to the first person of the Trinity, you emphasize the first note. *keyboard playing* If you're emphasizing the second person of the Trinity. *keyboard playing* And guess what the third person of the Trinity. *keyboard playing* And the motif for the devil, in the book of Revelation, there's also a [sic] Anti-Trinity. And so this is *keyboard playing* the devil. And there's the first person *keyboard playing* and the second. The Book of Revelation has about ten thousand words in Greek, almost exactly, and I didn't just go word by word because I want it to be good music.

I use this as the constraints to say what kind of good music is suggested by this pattern of motifs. And when I get to Chapter 21, the story talks about the New Jerusalem. Here's a Golden City coming out of the sky, and that's where I got to use my Prague theme. I'm not sure which… I'll just play it instead of trying to find the absolute best combination of voices. So it goes like this: *keyboard playing* So, da-da-da da-da-da da-da-da da. That's what I wrote down in the airplane. Then it goes on a few more bars, it uses some of the chord progressions of Dvořák's New World Symphony. And New World Symphony is like New Jerusalem. So this is a little bit of music here, that…

Zlatuška: So this is actually what we lose for not having international airport in Brno. Because if Don landed in Brno, that could have been Brno theme.

Knuth: So if I had never been invited here in the first place, who knows what would have happened. One other footnote to yesterday and that is: There was a question where I referred to Stuart Russell. Now I can give you the definite reference because I recommended that you watch this video. I think it's five minutes long, and it's easy to find. So Stuart Russell… *writing on blackboard* And there's been a lot of follow-up on it, but this came out not quite two years ago, and it's called Slaughterbots. It's sort of a mind-changing video that I recommend everybody to look at. It's very important, I think, that we should ban autonomous weapons, and there are thousands of computer scientists who have signed declarations and are actively trying to make sure that the dangers are minimized.

So that was end of footnotes to yesterday, now I'm ready for new question.

Zlatuška: So I guess… Have you ever burned out?

Knuth: There was a period about 1990, where I was feeling kind of low and I actually I got a little worried about it, because one of my uncle's had gone through a period where he was, I don't know, not getting along with anybody else in the family and so. So when he got old, and I said: "Oh-oh! Maybe I'm inheriting some mental problem." Anyway, so I went to see a psychiatrist, and he said: "Have you ever heard of a Type A personality?" He was a great doctor, he showed me his textbooks. Instead of telling me, sort of preaching to me, he said: "Here, look. Here's a book I was giving you. If you look in this chapter, you'll see something that describes you to a T (???)."

I learned from that how to reduce the stress, and so it was 1990. So how old am I? 50… 52 years old. And I realized why physical education was a required subject in college. I had never done much exercise, so I started swimming in 1990, and very quickly I was happy again.

But there was a time when… Well, I work sort of 24/7, in some sense, but it's fun. Mostly. I have to psych myself up in the morning, and once I get into something, then it's hard to stop again, so that's why I have to turn off like somebody mentioned yesterday. … Well, how do I describe my daily schedule?

I have a very peculiar scheduling algorithm. You wake up in the morning, and you have to decide what you're gonna do next. The algorithm that I finally decided to work the best is that I always do what I hate the most. Of all the things that I have no good excuse for not doing, which one would I rather not do. And that's what I choose. So all the time I'm working on something I don't wanna do, in a sense.

On the other hand, at the end of the week, I've got stuff out the door. And I'd have to do those unpleasant things anyway, so as long as there's no good reason to procrastinate anymore, that's what I choose to do. Somehow that turns out to be a better scheduling algorithm than the other way, where, "what is the most fun all the time?"

I noticed that my mood, when I start feeling bad, is really if several weeks have gone by, and I've never had a time to do anything creative. There's something, there's some urge that says: "Prove a theorem or something!" So if there's too much busywork, I can stand that for a little while, but after three weeks I have to try to do something new. I can't explain why that is. But that seems to be true.

Zlatuška: I would just add to that Stuart Russell, for anybody interested: There is a new book, which is Human Compatible by him and that was published yesterday.

Knuth: Oh, yesterday! *smiling* … Human…

Zlatuška: … compatible. Staying… Artificial Intelligence and the problem of control. And that looks pretty much as this topic.

Knuth: Very good.

Zlatuška: Tabs or spaces?

Knuth: Tabs or spaces? *smiling* You're going back to that question again? *Knuth laughs*

Zlatuška: Oh! Okay, it was… ???

Knuth: No, I have a question: Why does [sic] people ask about tabs or spaces? To me, when I make somebody else's code in order to read it, [and] it comes in with tabs, it confuses TEX. The Tab prints as a letter gamma, so I have to go through it, get rid of all the Tabs, and change to space, and then I can print the file.

Zlatuška: That's a proper answer, after which nobody would ever use Tabs.

Knuth: *laughing* Ok.

Zlatuška: Ok. Biggest challenge of becoming a good programmer?

Knuth: Biggest challenge of becoming a good programmer? Well, being born a geek. I know, some people think that it's just a matter of motivation and trying harder and keep educating, read and write right books and so on. Well, that might be true, but my experience is different.

My experience is that I know many many intelligent people who are, no matter how motivated they are, I don't think they'll ever be a good programmer, and conversely, I know that I'm never gonna be good as a programmer of a quantum computer. There's something about the way my brain works that gives me a great intuition for the classical computer, but not for quantum computing. And it's not a matter of good or bad or trying or harder or not being motivated. It's a quirk that I happen to [have] been born with one of these peculiarity [sic].

Zlatuška: What's your idea in this context about sort of mandatory courses of programming within elementary school? If it is true that not everybody can be a programmer.

Knuth: No, no! You left out the word "good". *laughter*

Zlatuška: Ah, ok. *laughter*

Knuth: I think people can become a programmer [sic], but dogs can walk on their hind legs.

Zlatuška: So you feel that there's not enough of bad code.

Knuth: What I'm saying is: If you somehow realize that you've been born a geek, then you owe it to the world to you use that talent because the world needs this talent. That's the way I look at it.

Zlatuška: Vim or Emacs… That was also yesterday. I am not quite sure…

Knuth: Vim or Emacs. Yeah, yeah, ok. I just did admit to using Emacs, but I learned how to use vi enough to know how to quit. *laughter* That was the hardest first lesson. I mean, I had to go Control-Q or something, I don't know.

Zlatuška: Do you think that we are living in a computer simulation?

Knuth: Do I? *smiling*

Zlatuška: Is your God a programmer?

Knuth: It's hard to tell. *laughter* So whether I think so or not doesn't matter. The whole question about how far Artificial Intelligence could go: It certainly could go to the point where it has decided that we should have this gathering today.

Zlatuška: There's one consequence of living in computer simulation because that means that the world would be only processes which are computable.

Knuth: And finite, yeah. So, for example, the game of life can simulate any such thing. You have any universal scheme, then it would represent the lecture I'm giving now. As one very special case, it would represent Stuart Russell's book, etc. It would represent all discordant ways to write the Fantasia Apocalyptica.

Zlatuška: What was the worst code you have ever seen?

Knuth: What was the worst code I have ever seen? Hey, I love this! *smiling*

Zlatuška: All you down to ??? level.

Knuth: No, no. I had this student in the 70s who was not born a geek *laughter*, but he came to Stanford from, I think, West Point. Anyway, he was trained as a soldier, and he would follow orders. He was not a Ph.D. student; he was master student. But he did a master's project which was to automate the system that we had in the Computer Science department for sharing technical reports with other universities. So we had to make subscriptions to others, and then if they send us their reports, we send them our reports gratis. We had this large database, sort of… data processing problem. So we needed somebody to do that, and I gave that to him as a master project. He wrote a system that was going to handle the technical parts. I saw the thing, I was busy at the time, and it looked like the right number of pages and so on. And it had a sample where he had taken a couple of… a small database with a few reports, and it gave the right report, so I gave him A on it, and he got his degree. Well, that was in June.

Then in July, the secretary called me: "So Don, we're having a little problem using this system." So I went, and that was one of my first trips to the Stanford AI Lab, where they had special computers there. I started looking at the code that he had written. I got to page three or page four, and it was an example of shellsort, a sorting routine, but it was implemented… It was the first time I had seen a program where I could change one character in the program and make it run hundred times faster. *laughter* And the thing was, the variable should have stepped by H instead of by one. So he wrote shellsort so that it was just a plain old insertion sort. And clearly, he didn't understand that. So I made a copy of that page: "Hey, I gotta show this to my students next ???!" Then I turned to the next page, and he did a binary search on that.

So every page I turned to, I saw a new kind of programming error that I had never seen before. *laughter* And finally I looked at the whole way he had organized the thing. It's really hard to explain, but the text editor that we used in that day—this was way before Emacs—it would start out with an index page, so text editor would always prepare automatically a table of contents at the beginning. He had assumed that text editor would always put all of the whole database into alphabetical order in just the way that you could read it, knowing that there would be line breaks or anything like this. So it was impossible to fix the program. You couldn't possibly write a program that was assuming that you were gonna use the text editor's database for table of contents to do the data processing. So that wins my vote.

Zlatuška: Well, given the fact that programming is actually about giving orders, and maybe he studied military academy. That means some privates should never be given the ability to issue orders.

Do you have any experience with psychedelic drugs, like Richard Feynman? Altered state of consciousness. LSD, psilocybin.

Knuth: No, I'm glad that I didn't, nor did my kids. As far as I know. Near-death experiences? No.

Zlatuška: Any problems you gave up on trying to solve?

Knuth: Any problems you gave up on trying to solve, yes. Many times, In fact, I certainly thought that I had proved that P was unequal NP rather earlier. And after I had solved it, then I wrote it up, and finally, everything disappeared, just about as I was writing the last line of the paper. I realized that it was hopeless.

I can remember the first time I actually solved a problem that I had given up on, which was a kind of a breakthrough. As a student, I had tried various things, and I sort of had a "give it five days and, if you haven't got any ideas, then let it go." But once, I let it go, and the next morning, I woke up, and I said: "Wait, what if I tried this." When you do work on a problem and fail, there's one trick you can try, and that is: Imagine somebody sent you email or letter or knocked on your door, and said "So ??? I just solved that problem." And then use: "Oh, I bet I know how he did it." Somehow, if you think the problem can be solved, sometimes it helps you solve it.

But the one that was most dramatic for me, I guess, years ago, when I started, about one third of Computer Science was a study of programming languages, and now my collected papers collected in eight volumes, and one of those volumes is all the papers that I wrote about programming languages. There's a journal called SIGACT News that has book reviews, and they have a page in there saying: "Here are books that we've received. Would you like to review them?" And this collection of my papers on programming languages was on that list for a couple years. Nobody wanted to review it. So nobody cares about this although this the hottest thing in Computer Science when I started.

I worked on a problem that was called parentheses languages. So I think everybody still knows what a context-free grammar is. A way of defining a language in terms of productions. But some context-free grammars have the property like nested parentheses. So if you look at the language, half of the characters are like left delimiters, and half of the characters are like right delimiters. And every string in the language has nested left and right delimiters, properly nested. So the question is: "Somebody gives you a grammar. Is the language that it generates a parenthesis language?"

The grammar won't show any matching between these, but can you somehow look at the grammar and figure out yes or no? Is it really gonna be possible to do this? That was an open problem, and I worked several weeks on it and finally gave up. But then, I think a week later, the key came to me how to solve it. And so, at that point, I was quite delighted to have the solution. I was able to use the insights I got from that when I was doing other work later on, but as far as I know, nobody has ever read that paper.

Zlatuška: Do you solve many programming problems or create music when you sleep?

Knuth: Do I… solve many programming problems, create music when I sleep? So, no. *laughter*

With respect to the Fantasia, it was kind of interesting that after I started working on it, I had the feeling that the music had already been written, and I just had to listen for it. I don't know, out of body experience… it's like I was channeling in a way, that it was there. It did feel that there was a muse helping somehow.

Now, with respect to programming problems, it goes the other way. If I'm trying to figure out how to solve a programming problem, I can't go to sleep. So I need somehow to either solve it or forget it. But when working on a difficult problem, I usually fill up sheets and sheets of scrap paper, and so I can't keep it all in my head, but I have to write down a whole bunch of stuff in order to get it into my brain. But when it comes to the point, when I can actually think about that problem when I'm swimming, then I know I'm about ready to solve. My brain has absorbed enough so that I've learned how to go from baby steps to giant steps in this territory, the problem domain. So there is this mysterious thing that takes place once I'm ready to do it.

So I always say to my grad students… They're working on the thesis, and I realized early on that was a good idea that they should keep. Every week when we would meet, they would write down in detail what they had been thinking about that week and what was on, what do we know, what don't we know. Then after the problem is solved, you can look at that, and I can use that to prove to them that they solved a hard problem. Because once you solve the hard problem, you think it was obvious, I didn't do anything. But once you see how many hurdles you actually got through… This was good.

Zlatuška: Also yesterday you mentioned that you are Platonist with respect to mathematics.

Knuth: Yes.

Zlatuška: Are you Platonist with respect to music? Music is up there and…

Knuth: The music of the spheres. There's a question. Why is it that some music I can hear ten times and next time I hear, it's just as if I never heard it at all.

Zlatuška: This happens to some students with mathematics. *laughter*

Knuth: With mathematics as well, yeah.

But, for example, in the old days, when you have CD, I mean long-play records, they would always have to add to the piece you wanted, they had to add something else to fill it out. So I have something by Brahms, and then the publisher added a piece by somebody named Bax. B-A-X. I've heard the piece by Bax as many times as I've heard the piece by Brahms because it's hard for me to get up and turn off the record player. However, I'll never recognize the piece by Bax the next time I hear it. So there's something different about Brahms's music and Bax's music, and I haven't been able to reverse-engineer that at all.

My piece [Fantasia] has parts of it that involve more less random elements. And the question was, well, if you just have random music, does the human brain somehow learn to do it? Well, it didn't work with Bax's music, but there are parts of it where I based on Morse code. I had to make a decision how to spell some Greek words. It turns out that in Ancient Greek, they had a notation for absolute pitch, so you could spell a Greek word just by playing those notes. Let's see if I can find… There's a place in Chapter 2 where the name Jezebel comes out. So Jezebel in Greek is iota, epsilon, zeta, and so on. I think it comes in here… Yeah. *music playing* That's Jezebel. It's kind of ???. Now Jezebel was a prophetess, and the motif for prophet is contrary motion. And so after I play Jezebel, then I play it contrary motion. But anyway, that's random. Still, our brain gets it in, we find ourselves humming. Jezebel, even though there was no reason why we should be able to remember that melody.

So I'm not sure to what extent you can take random elements, and they actually become warm and somehow have a personality. On the other hand, if you have no randomness whatsoever—musicians found long ago—that if you go straight one two three four, one two three four, exactly right, it loses life. You have to go a little bit before the beat, a little bit after the beat in order for music to come to life. And I tried the same experiment with font design. So instead of drawing a letter precisely, I would wiggle some of the points a little bit, and then the alphabet seemed to have a personality.

Zlatuška: Software patents. Software patents, can you elaborate your stance regarding software patents. Patenting software. The second from bottom.

Knuth: Second from bottom? Stances regarding software patterns.

Zlatuška: Patents. Intellectual property.

Knuth: Patents! Ok, aha. Software patterns is another thing, okay software patents, right.

I think people deserve protection for their ideas, but not if just the ideas are trivial. So a great number of software patents were something that we would expect any student to do on an exam, but a lawyer—a patent lawyer not being a geek—wouldn't know how to distinguish those. And so there was a time when people went and tried pro bono make a patent on every trivial idea. So that people wouldn't be able to make us pay them every time they use this trivial idea.

When I wrote TEX, I didn't need to get permission for any of the ideas that I use, the trivial ideas that I used in TEX. But after intellectual property rights got more and more complicated, it might very well have been impossible to write TEX twenty years later.

So when it comes to a substantial piece of software, like the undoing mechanism in Photoshop or something like this, I think this really deserves patent protection for a limited amount of time, but not in perpetuity. That's my general feeling about patents in a nutshell.

Zlatuška: What's your favorite music band?

Knuth: My favorite music… band? *laughter* I remember The Beatles. But lot of the music after that sounds like noise to me.

Zlatuška: Some of the exercises in The Art of Computer Programming are known to be open research problems. Has anyone ever contacted you that they have solved one of them while reading the book?

Knuth: Yes, I think I mentioned that yesterday, but it's not that uncommon.

Zlatuška: And you don't pay actually…

Knuth: No, no, I only pay…

Zlatuška: It became closed problem, so it was an error…

Knuth: It's stated there that if you find a better answer, you don't get money, you get glory instead, and so instead I mention your name. But if you correct an error, I silently correct it as if I had known it.

Zlatuška: What would you like to redo?

Knuth: What would I like to redo if I could? I would base TEX on decimal instead of binary at the lowest level. TEX is based—at the lowest level—on a scaled point, of which there are 2^{16} scale point to weigh a point. So internally, everything is kept in binary but is communicated to the user in decimal. So the user doesn't get to see… This leads to strange results. You know, you say one third, and then you multiply it by three, and you get 0.999 instead of one. So that would have been better to do that.

Zlatuška: Do you still use TEX often?

Knuth: *giggles* Well, let's see… I haven't used it since last Friday because I've been in other town.

Here's a… Oh, I see. You're not raising your hand, you're raising the camera. Ok. But other people out here who…

Zlatuška: Anybody from the audience?

Someone: Can you play something?

Zlatuška: Can you play something?

Someone: Like a longer piece.

Zlatuška: Longer…

Someone: Anything you like.

Knuth: Well, I only have this music here. So, choose a random chapter.

Someone: Uh… Seven? *laughter*

Knuth: Seven. Ok, so seven is the chapter where the saints come marching in. And…

Zlatuška: 3:16?

Knuth: What?

Zlatuška: 3:16?

Knuth: So there are different motifs here. I'm trying to see which are the easiest to explain. Well, ??? mentioned that [it] has lots of different styles, and since this is about the Apocalypse, one of the styles had to be calypso. So in Chapter 7, we get calypso: *music playing* Now, that's trying to imitate an organ. Let's try to just do a piano. I don't know how to do this here. Voice… Voice "church [organ]"… Let's change other voices… How do we go down?

Szaniszlo: What kind of voice?

Knuth: Just take piano, for example. Grand piano, here we go. *music playing* Now this last thing is *music playing* Harry Belafonte, so "run Venezuela". *music playing and Knuth singing* "She ran with the tailor." *laughter* That's what I'm singing to myself when I was writing this piece. However, this is, of course, taking place as the saints come marching in. *music playing* So at this point there's a grand shout, comes along, and *chord plays* at the is called *music playing*. That's God. *music playing* Next is: *music playing* This is the motif for the lamb, one of the principal characters, and it's supposed to sound like baa baa. *music playing* So the scene we have here is that the saints go onto this hill, and then there are 24 elders, and the elders form a chromatic scale of 24 notes. *music playing* Besides the elders and other characters are Seraphim, and there are four creatures, and the theme for the four creatures is *music playing*. And you can do this: *music playing*.

Is that enough? *applause*

Zlatuška: By the way, is it difficult to actually play something for organ just on piano? If I understand…

Knuth: Yes, but it's also very difficult… It's also very difficult to play this on the organ. Jan [Rotrekl, the performer of the Fantasia Apocalyptica], the organist, has had to work very hard in order to do it. When I wrote this piece, I didn't hold back. I knew it was the only piece I was ever going to write, and so I didn't bother to simplify it. I wrote what I thought I wanted to hear, and so he has to play what I wrote.

Zlatuška: With (???) Don, when I asked him [Don] whether we can perform this, he mentioned that the organist who did the world premiere was ill and unable to play that, but there is a proof that it is playable. *laughter*

Knuth: *laughs* Yes. I can play it, but not at speed. Well, the organist who did work on it actually fell in love with—I'm glad to say—and he wrote me a couple weeks ago saying that you he's feeling withdrawal symptoms, he wants to play it again.

Zlatuška: Anybody else?

Someone: I have a question. I have read about your WEB and CWEB projects. Do you think that the problem in Computer Science that you were trying to solve with this projects is already solved by maybe new programming languages or different approaches towards documentation? Or do you think that the projects are still relevant today? And maybe a follow-up: If you had ability to write CWEB or WEB again today, would you do anything differently than you did before?

Knuth: To me, it's one of the things I love the most about my life, is that I can write programs in CWEB. However, in order to do that, I'm also living with the fact that the implementation that I have had never been tuned up to a great programming environment. So, for example, I start a CWEB program, and I always type a few things. Like I always type a line that says: |@...| at-sign asterisk index period, and I put that at the bottom. It's a little bit of a nuisance, but in fact everything in life has some small nuisances, and so as long as something is working for 97 % of the time, I'm not gonna spend much time figuring out the 3 %, but a mature program, they start fixing up the 3 %.

So I use CWEB knowing that it was a quick and dirty implementation, but it still does almost everything that I want, exactly as I want. And I have an Emacs interface to CWEB. My wife will tell you I come out of my office several times a day saying: "It's so fun programming in CWEB!"

And the reason is that as I'm doing it, it seems to me that I'm combining the formal and the informal aspects of the program the way they ought to be. In order to understand any complicated technical subject, one of the main tricks of it, of a person who writes about Computer Science, mathematics, is to say everything twice: once formally and once informally. Maybe three times, but from different perspectives, but you don't only give a formula, but you say what the variables in the formula mean. You write a program, you not only declare something to be ??? and a variable. You

say what has an invariant relation: This is the number of nonzero elements in the array or something, but you combine formal and informal.

And that's the way CWEB works. It breaks down a program into a small number of pieces that fit together in a small number of ways. But each module of the program is a combination of informal—which you write in natural language—and formal—which you write in whatever formal language you're using. In CWEB, it's C, but there are many different flavors of WEB.

So I really think there's nothing else anywhere near as good as my approach, but I know that there are many ways to refine it, and I'm not interested in actually pursuing the refinements because it's good enough for me. On the other hand, most of the world writes… you look on the solid programs on GitHub. You'll find that they're probably not using CWEB, but there's a certain style that's become… I don't know, I don't like C++ program because… it's so ambiguous. Each C++ compiler does different things with these programs, and so when people… If somebody sends me a program in C++, I don't know what subset of the language they're using. There's all kind of things going on automatically, and the programmer knows what she's doing, but their reader doesn't know which subset is there, so I don't care for that.

But let's suppose we look at a typical module that we get, that's written in C, and it's a style of programming that people have learned to read and maintain, and so it works. I can say: "Oh yeah, let me rewrite your program in CWEB, and you'll see that it actually not only is better, it's more reliable, and you'll realize that certain features were missing because of the discipline of literate programming." However, I don't believe I'll ever convert the world with this. It'd be like saying Esperanto is a much better language than English, so therefore let's abolish English. English works well enough so that some ideal language isn't going to displace it.

Zlatuška: There's question by Tim: Paul Erdős spoke of book in which God kept the most elegant mathematical proofs in the same sense of divinity that are key??? I would extend this also to the most elegant programs, but…

Knuth: I read a very strange paper recently where they asked people to compare algorithms to music. One group of subjects, they were supposed to compare algorithms to music, and another group of subjects was supposed to compare the algorithms to art. Where was this paper? It was one of the papers I read in the last three weeks. Anyway, these people presented keep sort, binary search, … they took five algorithms that they thought were classic, and then they showed the subjects several pieces of music, and the subjects were supposed to say: "Okay, now match this algorithm to this piece of music." They found—maybe a hundred subjects—that there was a fairly good correlation, it wasn't random permutation of this matching between algorithms and music. But then they did another group, and they showed them five paintings, and they were supposed to again associate this with the painting, and that didn't work at all. But maybe they didn't choose the right paintings. I don't think all kinds of art are equivalent. Certainly, there's a quantitativeness to music that's similar to Computer Science.

Zlatuška: Suppose we create a technology to faithfully copy and simulate working human brain. Would you want to continue living as such a simulation? That's apparently the Kurzweil's idea.

Knuth: So there's a line in… That's a Gershwin's song Ain't Necessarily So. And it says something like Methuselah lived nine hundred years. Methuselah lived nine hundred years, but who calls that livin' if no gal won't give in to a man who was nine hundred years. So there's more to living than thinking.

Knuth: Your favorite fractal?

Knuth: My favorite fractal. Well, it has to be the dragon curve because that was the first one I knew about. The dragon curve—you can look it up—but basically, you take a long sheet of paper. Like thin sheet of paper, like we used to have adding machine tape or something. And you fold it in half, you crease it, you fold it again, and you fold it again, so each time it gets half as big as before. And you've got creases in the paper. Then you open it up, some of the creases go down, and some of them go up. It makes a pattern. An interesting pattern that also turns out to be equivalent to the Legendre symbol of minus one over anything (???). Anyway it's a pattern. You open up the paper, and you make all the bends go ninety degrees. So it'll go like this *drawing on blackboard*, and then go like this… something, left-right, left-right. You can round off the corners if you want, but this is the idea of dragon curve.

It turns out it never intersects itself, and if you take four of them, and start them out… I probably didn't drop correct dragon curve, but if I take four of them, it will cover the entire plane, with four of them. I had a lot of fun proving that theorem in the 80s. And I was really proud when that theorem was picked up in Russia in Quant magazine, which late sixties, of course, there was the iron curtain then, and they illustrated my theorem with four colors in this magazine written for high school students in Russia. It says, you know, I knew enough Russian to translate it, it said: This is a difficult theorem, it was proved by Donald Knuth.

Now we see Czech version of… my books.

Zlatuška: I think the topmost question was already tackled.

Knuth: Are there any other questions from the…

Zlatuška: From the floor?

Someone: What kind of organ do you have at home? Is it a pipe organ? Mechanically controlled or a digital or electronic…

Knuth: Yes, it has 850 pipes or so, and the website explains it all. If you go to my home page and look under pipe organ. It was built by a firm called Abbott and Sieker—they're both dead now—but they used to make about four organs a year. And it has 17 ranks of pipes, and I have a major exercise in Fascicle 5—which is coming out next month to say—how many different sounds can you make on this organ that have exactly five pipes going, or exactly six pipes going, or so on. I found it to be quite a fascinating exercise. So if you want to know more about the organ, online has the specs, but also then you've got to buy the book, [to] look at this exercise about that organ that I have.

Someone: So I assume that you also prefer pipe organ, mechanically controlled, instead of say, digital organs.

Knuth: I am sorry, I don't understand the question.

Zlatuška: You prefer classical organ to digital?

Knuth: Oh, I see. Yes, absolutely. I heard a pretty good digital organ in England in July, but it's quite rare. They make good digital recordings of organs, most of the organs that I've ever had a chance to play. Even though it's in a great building, and you had the reverberation and everything, it just doesn't match.

I come from a part of the United States where it was illegal to some—it's called America's Dairy State, America's Dairyland—and so it was illegal to sell margarine instead of butter in our state. If you wanted to get margarine, you had to go to Illinois *laughter* to buy it, it was a little cheaper. But I look at butter versus margarine, that sort of has the difference between pipe organ and electronic organ. But also in the early days, before we had good resolution in fonts, there was good printing–butter–and there was the kind that we could get in our lab in the early days–margarine.

Zlatuška: You originally wanted not to have your organ built by some American company, but you imported from some Nordic country. Do you regret that did not work? Or…

Knuth: I heard some really beautiful organs in Denmark, and I inquired about having a Danish organ, and this was in the 70s, there was only one Danish organ in America at that time, it was one in Boston. So I had talked to the builder, and then I found out that there was no way… I had a limited budget, and according to Danish law, the only way I could buy this organ was to agree that the price was indeterminate until after it was built because by Danish law whatever the Union workers wanted to get for it, had to be the amount that was done. So they couldn't get me a fixed price for it, and I might have gone broke, so I was unable to do that.

Zlatuška: So your love for organ was not that big that you would get broke because of it? *laughs*

Knuth: The organ that I finally bought cost $35,000. People were paying that for a house in those days. Now you get a house for $35,000, it's impossible.

Zlatuška: What will happen with your organ when the lease of your house expires?

Knuth: Who knows. But it has only existed in this room. People could unscrew it and reassemble it somewhere.

Knuth: Somebody else? Ah, over there.

Someone: You mentioned once that in Super Bowl game the crowds, they have flags with… showing 3:16 on it. I've always been puzzled what does that have to do with football game.

Knuth: There are people who flaunt their religion. And the most famous verse in the Bible by its number is John 3:16, which is said to be the gospel in a nutshell. It says the God loved the world in such a way that He gave His only child, and so on. So that's a very famous verse of the Bible. People think that by putting that number up that will give publicity, to make people look up this verse and they will suddenly realize that this should be their religion. At football games, it just happened that the cameras, the camera crews survey the crowd, and so this was a way to get advertising for whatever slogans you wanted to do. And a certain group of people started doing it.

Then there were jokes based on. I mean, there was in baseball game… what was his name… a guy from the Boston Red Sox, but anyway his batting average was 0.316, and his nick, his first name was John. So people hold up a chart saying "John! 0.316!" And this was to be a satire on this phenomenon. But it was something that sort of grew like… we have bumper stickers, slogans that people put on their cars and things like that.

But the fact is that this number, the verse got popular by its number, and I used that later when I wrote this book called 3:16 because I wanted to use a cross section of the Bible in order to understand the complexity of it and have some way of sampling. So I used this in order to get a good cross section of not the Bible itself so much, but all the secondary literature about the Bible. So there have been 100,000s of books written about the Bible, but I could go into a theological library, and most of those books have an index in the back, and they'll say which verses do I refer to. So I go through, and I find out, oh yeah, I only have to read a dozen pages of this book, and I can see what it says about Genesis 3:16 and what it says about Revelation 3:16.

By the way, Revelation 3:16 is one of the verses that's here [in Fantasia Apocalyptica], so I might as well go to Chapter 3. *flipping pages* The verse Revelation 3:16 says something like "because you are lukewarm, neither cold nor hot, I will spit you out of my mouth." The message is that God prefers atheist to people who don't care at all. So when I do verse 3:16, I had a little fun in the music. I used time signature 3/16, which is three sixteenth notes to a measure. And then it says: "I will spit you out of my mouth," so on the organ, we have here in the church on Friday, has a pipe called the spitzflute, so we use a spitzflute for this spitting out of the mouth. So it goes anyway *music playing* and then spitz! *music chord* Listen for that on Friday.

Zlatuška: I guess that we basically finished the time allocated for this. There will be the question: "How are you today?" but that will be postponed for next time. Maybe if Don wakes up in the night, maybe he starts with another sermon in the Church, but… Now, that was just a joke.

So thank you very much, I believe that… *applause* Thank you very much for coming, for having these sessions with us. An invitation to everybody for Friday, 7 p.m. in the Church of Jesuits, where the piece Fantasia Apocalypsa [sic] will be performed.

References

KNUTH, Donald, 2020a. *Czech Premiere of Donald Knuth's Fantasia Apocalyptica* [online]. Ed. by NOVOTNÝ, Vít; MAČEJOVSKÝ, Šimon. YouTube [visited on 2020-10-08]. Available from: https://youtu.be/wk7dEKMPP68.

KNUTH, Donald, 2020b. *Fantasia Apocalyptica* [online]. Stanford Univerzity [visited on 2020-06-23]. Available from: http://www-cs-faculty.stanford.edu/~knuth/fant.html.

LUPTÁK, Dávid, 2019. Fantasia Apocalyptica: Česká premiéra. *Zpravodaj Československého sdružení uživatelů TEXu*. Vol. 29, no. 1–4, pp. 11–18. ISSN 1213-8185. Available from DOI: 10.5300/2019-1-4/11.

SOJKA, Petr, 2019a. *Donald Knuth a Dana Scott: Týden s držiteli Turingovy ceny v Brně* [online]. Ed. by KUBÍČEK, Petr. Faculty of Informatics, Masaryk University [visited on 2020-06-23]. Available from: https://fi.muni.cz/events/2019-10-donald-knuth-dana-scott-turing-prize-laureates-brno.html.

SOJKA, Petr, 2019b. *Otázky a odpovědi s Donaldem Knuthem: Umění programování* [online]. Ed. by KUBÍČEK, Petr. Faculty of Informatics, Masaryk University [visited on 2020-06-23]. Available from: https://fi.muni.cz/events/2019-10-08-donald-knuth-question-answer-session-computer-programming-as-an-art-brno.html.

SOJKA, Petr, 2019c. *Otázky a odpovědi s Donaldem Knuthem: Zájmy bez hranic* [online]. Ed. by KUBÍČEK, Petr. Faculty of Informatics, Masaryk University [visited on 2020-06-23]. Available from: https://fi.muni.cz/events/2019-10-09-donald-knuth-question-answer-session-boundless-interests-brno.html.

SZANISZLO, Tomáš, 2019a. *Donald E. Knuth Q&A Session 1 Transcript* [online]. Faculty of Informatics, Masaryk University [visited on 2020-06-23]. Available from: https://fi.muni.cz/events/2019-10-08-donald-knuth-question-answer-session-computer-programming-as-an-art-transcript-brno.html.

SZANISZLO, Tomáš, 2019b. *Donald E. Knuth Q&A Session 2 Transcript* [online]. Faculty of Informatics, Masaryk University [visited on 2020-06-23]. Available from: https://fi.muni.cz/events/2019-10-09-donald-knuth-question-answer-session-boundless-interests-transcript-brno.html.

ZLATUŠKA, Jiří, 1996. *Donald E. Knuth doktorem honoris causa Masarykovy univerzity* [online]. Institute of Computer Science, Masaryk University [visited on 2020-06-23]. Available from: http://webserver.ics.muni.cz/zpravodaj/articles/59.html.

Tomáš Szaniszlo
xszanisz@fi.muni.cz

Translations from a Vocabulary

Handling translation into various languages

Abstract
Formerly part of the module hvdm-xml but now split off into an independent module with its own description. Used for making other modules language sensitive. The module is especially tailored for XML use.

Introduction

Elements of the output can be internationalized through definition and use of one or more vocabularies. This module allows for a flexible and adaptable translation of individual words. The component effectuating this is the module `hvdm-voc`. Its interface is meant to be accessed from XML as well as through ConTEXt-macros.
This module certainly is *not* a full blown translator. Its scope is restricted to the translation of individual words, not even plural forms are automatic and must be added separately. But although simplistic in nature, it provides for the automatic adaptation of certain keywords to a change of language.

It all started with the construction of a database in XML format with historical facts about my ancestors. Each fact resides in a separate file to be processed by my note processing module `hvdm-tak`. Everything is enclosed in an XML node with suitable node names chosen. Since the lingua franca in computing is the English language, it seemed natural to use this for these names. Thus each person in my notes became to be described by a `<person>` node, for instance:

```
<note>
    <person>
        <name>Arnoldus van der Meer</name>
        <age>28</age>
        <profession>seller of mineral water</profession>
    </person>
    <!– other nodes –>
</note>
```

Typeset in ConTEXt with `\language[en]` such a note looks as in the next figure. Note the english keywords at the left edge corresponding to the node names. The remainder of the text is in Dutch, as might be expected for material taken from five centuries of dutch archives.

note-1 — file: events/hga-0402-422-1-99.xml

subject: Schuldinning door Arnoldus van der Meer
date: 5 November 1779
category: event/weeskamer
key: vandermeer blauwsonnevelt prins
author: Hans van der Meer
source: Haags Gemeentearchief 0402-01 inv.422 pdf-1 p
location: Den Haag
person: Arnoldus van der Meer
role: schuldinner
person: Francina Prins
relation: weduwe van Frederik van Blauwsonnevelt
role: erfgename
person: Frederik van Blauwsonnevelt (Fredrik van Blaau
relation: overleden van echtgenoot Francina Prins
abstract:
Arnoldus van der Meer neemt op zich om voor de Blauwsonnevelt en hun twee minderjarige kindere van f325–15 te innen.

Language set to English.

Not all members in my family did like the english words interspersed between the text in Dutch. Thus arose the idea for this translator. The keywords with their translation were put in a vocabulary as described below. In the notes module their typesetting is enclosed in a `\translate` macro call. The result is shown in the second figure, again note the keywords at the left edge.

notitie-1 — file: events/hga-0402-422-1-99.xml

onderwerp: Schuldinning door Arnoldus van der Meer
datum: 5 november 1779
categorie: event/weeskamer
sleutel: vandermeer blauwsonnevelt prins
auteur: Hans van der Meer
bron: Haags Gemeentearchief 0402-01 inv.422 pdf-1 p.
plaats: Den Haag
persoon: Arnoldus van der Meer
rol: schuldinner
persoon: Francina Prins
relatie: weduwe van Frederik van Blauwsonnevelt
rol: erfgename
persoon: Frederik van Blauwsonnevelt (Fredrik van Blaa
relatie: overleden van echtgenoot Francina Prins
samenvatting:
Arnoldus van der Meer neemt op zich om voor de Blauwsonnevelt en hun twee minderjarige kindere van f325–15 te innen.

Language set to Dutch.

But (inevitably) I became a bit sloppy. Dutch words crept in as node names for new properties which were added in the course of the genealogical investigations:

```
<note>
    <person>
        <name>...</name>
    </person>
    <samenvatting>an abstract</samenvatting>
</note>
```

The following figure shows what happens if the note is typeset in the english language again. That one keyword *samenvatting* strangely deviates from the others.

person: Frederik van Blauwsonnevelt (Fredrik van Blaau
relation: overleden van echtgenoot Francina Prins
samenvatting: Arnoldus van der Meer neemt op zich om
erik van Blauwsonnevelt en hun twee minderjarige kind
taal van f325-15 te innen.

Language set to English with Dutch node.

Although changing node names to their english equivalents is a simple action in an editor, it provided the incentive to make the translator module a bit more general. Instead of translating from one language only, it should allow translation between any two languages in the vocabulary. Look at the fourth figure where this flexibility is demonstrated by changing to `\language[de]`.

—— *Notiz-1 — Datei: events/hga-0402-422-1-99.xml* ——
Thema: Schuldinning door Arnoldus van der Meer
Datum: 5 November 1779
Kategorie: event/weeskamer
Schlüssel: vandermeer blauwsonnevelt prins
Autor: Hans van der Meer
Quelle: Haags Gemeentearchief 0402-01 inv.422 pdf-1 p
Ort: Den Haag
Person: Arnoldus van der Meer
Rolle: schuldinner
Person: Francina Prins
Beziehung: weduwe van Frederik van Blauwsonnevel
Rolle: erfgename
Person: Frederik van Blauwsonnevelt (Fredrik van Blaau
Beziehung: overleden van echtgenoot Francina Prins
Zusammenfassung:
Arnoldus van der Meer neemt op zich om voor de
Blauwsonnevelt en hun twee minderjarige kindere
van f325-15 te innen.

Language set to German.

XML Interface

Everything starts with the creation and filling of a vocabulary. By default the module provides a vocabulary to which one can add translations, but it is possible to create others as has been done in the code below. A new vocabulary is created the first time its name appears on a `<vocabulary name="name">` node, further such calls with that name are silently ignored. Creation of a vocabulary will *not* make it automatically the current vocabulary.

That should be done by the separate `set` attribute as in the example below.

```
<vocabulary name="myvocab" set="myvocab">
    <word>
        <en>dutch</en>
        <nl>nederlands</nl>
        <de>niederländisch</de>
        <fr>néerlandais</fr>
    </word>
</vocabulary>
```

Load the above data from a buffer or a file with:

```
<vocabulary buffer="aBuffer"/>
<vocabulary file="aFile"/>
```

Vocabularies are switched with the `set` attribute. A value 'default' for the name performs a switch back to the default vocabulary installed by the module.[1] The code below illustrates how to set and retrieve the names of the current vocabulary and language.

```
<vocabulary set="myvocab"/>
<vocabulary show="vocabulary"/>
<vocabulary show="language"/>
```

The results are vocabulary = *myvocab* and language = *en*. Note that attributes `name` and `set` behave differently. When the named vocabulary does not yet exists the former will create a vocabulary with that name, whereas the latter will issue an error message instead. When both attributes are present the vocabulary is created first and then made current.

Individual translations can be added to any named vocabulary, but when there is no `name` attribute on `<vocabulary>` the current vocabulary will be extended.

```
<vocabulary>
    <word>
        <en>greek</en>
        <nl>grieks</nl>
        <de>griechisch</de>
        <fr>grec</fr>
    </word>
</vocabulary>
```

With the current language being *en* this results in greek, Greek and GREEK. Changing the language setting to german with

```
<vocabulary use="de"/>
```

will change to griechisch, Griechisch and GRIECHISCH.

Translations are retrieved by a `<vocabulary>` node with attributes `get`, `Get` and `GET`. The three variants select the corresponding case variants. Presence of a `use` attribute translates into that language but leaves the current lan-

guage setting unchanged. For instance:

```
<vocabulary get="greek"/>
<vocabulary use="nl" Get="dutch"/>
<vocabulary use="fr" GET="english"/>
```

produce griechisch, Nederlands and ANGLAIS. The presence of a 'get'-ter forces the change from attribute `use` to be locally confined. Although the last language accessed here was from `use="fr"`, the current language *de* has not changed.

The vocabulary is set up in such a way that translations between all language pairs are possible. In itself that sounds nice, but what if a synonym has to be added? For example, besides 'dutch' translated into 'niederländisch', we want the twoletter code 'nl' to be translated into 'niederländisch' too.
The problem here is the following. Addition of 'nl' in the same manner as demonstrated above, will overwrite cross translations already present instead of merely adding the equivalents for 'nl'. The solution is simple. Use `<word add="nl">` and the enclosed translations will be taken for synonyms. In this manner 'nl' is added to the vocabulary without generating cross translations. We will find for instance 'nl' translating into *niederländisch* just as happens when translating 'dutch' to german.

```
<word add="nl">
    <en>dutch</en>
    <nl>nederlands</nl>
    <de>niederländisch</de>
    <fr>néerlandais</fr>
</word>
```

A problem still remains with this translation scheme. Let us add translations for icelandic, switch to dutch and see what `get` and `Get` translations do: *ijslands* and *Ijslands*. The latter is wrong because in dutch the 'ij' counts for one letter! Thus 'Ijslands' should have been 'IJslands'. Luckily the solution is not problematic. Add an extra node for ijsland → IJsland as a synonym with an adapted language code `Nl` The first letter of `nl` now being in uppercase. Instead of raising the first letter of the translation only, the translator then uses the alternative definition:

```
<Nl>IJslands</Nl>
```

With this addition to the vocabulary we now get *IJslands* as it should be. The same problem arises for letters such as the ç in français leading to FRANçAIS instead of FRANÇAIS or the ä in Niederländisch. Here we could have solved it with other exceptions as `<FR>FRANÇAIS</FR>`, but instead the upper-lowercase translator has been made a little bit smarter. It knows how to do a case change for letters like é, à, ü, ç.

ConTEXt Interface

Although primarily developed for use in an XML environment, it boils down to calling into TEX code. It is therefore always possible to fall back onto the underlying macros. The following are the API calls available.

- `\VocabularyCreate[#1]` – creates a vocabulary named in `#1` if it does not yet exists, otherwise do nothing. Note that the current vocabulary is not changed, that should be done explicitly with the set macro.
- `\VocabularyDelete[#1]` – use this in the rare case one wishes to get rid of a vocabulary. The default and the current vocabulary cannot be deleted. Nor is it possible to remove items from a vocabulary once they have been added.
- `\VocabularySet[#1]` – the vocabulary named `#1` will be made the current one. Reset to the module's default vocabulary by calling with an empty parameter.
- `\Vocabulary` – the name of the current vocabulary. Example: the current vocabulary is *myvocab*.
- `\VocabularySetLanguage[#1]` – the two-letter language code makes it the current vocabulary language.[2] An empty argument will set it to the value of `\currentlanguage`.
- `\VocabularyLanguage` – retrieves the two-letter language code of the current language. Example: after changing to french by calling `\VocabularySetLanguage[fr]` the current language at this point is *fr*.
- `\VocabularyLoadFromBuffer[#1]` `\VocabularyLoadFromString[#1]` `\VocabularyLoadFromFile[#1]` – these macros load data as XML nodes from string, buffer and file. Example: prepare with `\startbuffer[spanish]` a buffer to add spanish:

```
<vocabulary>
    <word>
        <en>spanish</en>
        <nl>spaans</nl>
        <de>spanisch</de>
        <fr>espagnol</fr>
    </word>
</vocabulary>
```

and load it with `\VocabularyLoadFromBuffer[spanish]`. Now translation of 'spanish' in *fr* will be *espagnol*. Similarly use `\VocabularyLoadFromFile[italian.xml]` from the prepared file `italian.xml` and obtain *italien* for 'italian'.

▷ `\translate{#1}`
`\Translate{#1}`
`\TRANSLATE{#1}` – translate their argument into the current language with no case change, first letter uppercase, all letters uppercase, respectively. Example: `\TRANSLATE{dutch}` in the current language *fr* results in *NÉERLANDAIS*. But an absent translation returns its argument unchanged as in `\Translate{japanese}` is *Japanese*.

▷ `\VocabularySetLanguageDefault[#1]`
`\VocabularyLanguageDefault`
`\translateDefault[#1]` – There are situations where one language is special. An example is found in the module mentioned in the introduction. Nodes `<author>`, `<person>`, etc. need special treatment in the program. This is accomplished by attaching a flag. Without a common default language it would have been necessary to flag all occcurrences of `<author>`, `<auteur>`, `<Autor>`, etc. separately. By using `\translateDefault` this can be avoided because it enables the programmer to collect all occurrences into a common language. An empty argument `\VocabularySetLanguageDefault[]` sets to the value of `\currentlanguage`. Note that setting of the default language is done globally.

Availability

The module and its supporting modules can be downloaded from my site `hvandermeer.com/publications.html`. The TEX-stuff resides in section "Articles on TeX", downloads are a little below in the link "ConTeXt module distribution".

Notes

1. Note that a vocabulary named 'default' cannot be used and will raise an error if one tries to do so.
2. Do not be tempted to try 'Fr' or 'FR' because the module will silently convert both to 'fr'.

Hans van der Meer
`havdmeer@ziggo.nl`

Macros and Lua snippets

Helper macros mostly in Lua

Abstract
Described is a module containing a number of helper macros, many of them programmed in Lua.

Keywords
Lua, macro, file, font, list, date

Introduction

As already mentioned in a previous article in the MAPS, I am somewhat of a do-it-yourself'er. Either because the macro needed was not present, not on my radar or just because it seemed something nice to have and possibly useful in the future. These macros may be of use to others but it is up to them to decide that, of course. I just hope some might solve a problem here or there. They can be downloaded from my website at `https://www.hvandermeer.com`. From the homepage jump to the page *Publikaties and TeX modules* where they can be found as *ConTeXt module distribution*.

This module named `hvdm-ctx` is dependent on the companying module `hvdm-lua`. The latter contains most of the implementations of the caller macros in the former. The macros can be loosely categorized as file related macros and macros for the creation, maintenance and querying of Lua tables, as well as date and string manipulations plus some macros without much relation to each other.

Files in a Filevault

The Files group of macros is for manipulating file names, file content, combining files and probing the internet.

The idea for the *FileVault* macros arose from my use of XML for collecting in small notes the data found in various 17th and 18th century archives. This collection is nearing a thousand XML files with occasionally the addition of new ones or updated content. Some people will check the correctness of the XML before submitting their files, but that's not one of my habits. Therefore I decided to check them for validity in processing runs using the `XMLCheck` macro described below. But this implied reading them twice, once for the check and once for typesetting.

Why not read them once and keep the content as a string in Lua's memory? Modern memories have Gigabyte size and my thousand XML files are peanuts. I could have relied on the file caching of the OS, but it is not in my nature to forgo a chance to program something nice. Thus a small set of macros was developed to 'read once, use anywhere' so to speak. Use of this should be completely transparent, naturally, not needing special treatment. Switching between in memory content or rereading from disk to be done by simply turning it on and off. A bonus will be the possibility to run filters on the input before handing over to ConTEXt.

The above "not needing special treatment" could not be held on to entirely, pity. The culprit turned out to be the backslash. My notes can contain `<tex>` nodes, escapes to typesetting part of the text directly with ConTEXt. Typesetting files with for example `\loop ... \repeat` statements inevitable leads to a crash. In the underlying Lua code backslashes have a special meaning as escape string for control characters newline `\n`, `\t` or `\xxx` where xxx is a three digit number. TEX's use of the backslash doesn't fit in this scheme and things like `\def` fail miserably.
For possible solutions one can think of replacing `\` with its escaped equivalent `\\` but then they turned out to be neglected completely. Trickery with catcode changes is not encouraged in general and often extremely dangerous and disastrous.
Sad, but complete transparancy had to be relaxed a bit in order to cope with this situation. Its solution required the implementation of a macro that will leave the content of the file in an intermediate buffer in ConTEXt. Running code like

```
\xmlprocessbuffer{}{buffer-containing-file}{}
```

is then possible without causing a crash.

These are the functions implemented for the FileVault:

▷ `\FileVaultGetState`
`\FileVaultSetState[#1]` – handles the current state of the FileVault. The first macro returns the current state of the FileVault. In accordance with ConTEXt

custom this is `start` or `stop` whether the vault is active or inactive. Activation and deactivation is done with the second macro. The option values are `[state=start]` and `[state=stop]` respectively.
When the vault is in an inactive state all actions except these state macros and those reading files are suspended (see the example below for the latter). The latter will either return an error message or silently do nothing.
Examples: `\FileVaultGetState` shows the initial state of the vault as *stop*. After changing it with `\FileVaultSetState[state=start]` we can verify that the state is *start*.

▷ `\FileVaultFileString{#1}` – returns the file content as a string. If the file is nonexistent the macro silently returns an empty string.
Example: `\FileVaultFileString{example.txt}`

This is a very small example file.

What will happen if the vault is inactive? The same statement is executed after stopping the vault. The state is now *stop* and the same example gives

This is a very small example file.

Thus showing the transparency of the vault system for file reading.

▷ `\FileVaultFileBuffer[#1]{#2}` `\FileVaultFileDefaultBuffer{#1}` – returns the content of a file put into a ConTEXt-buffer. The parameter between braces `{}` specifies the file, the parameter between the option brackets `[]` must be the name of an existing buffer. The return of both macros will be the name of the buffer where the file content has been stored. With an explicite buffer name the contents of that buffer stays available, otherwise the internal buffer is reused each time. If the file is nonexistent the macro silently returns an empty buffer.
Example: `\typebuffer[...{otherexample.txt}]` results in

```
This is the other example file.
```

Since the underlying action is either file reading or retrieval, this action is transparent to the state of the filevault as for `\FileVaultFileString` has been shown above.

▷ `\FileVaultList` `\FileVaultListWithCount` – shows the content of the FileVault, the second caller adds the number of times each file has been accessed. Between the filenames in the list `\crlf`'s are added.
Example: After reading the second file twice again, the current access counts are:

example.txt 1
otherexample.txt 3

An inactive vault will return the following appropriate return:

ERROR inactive filevault

▷ `\FileVaultReturnTransform[#1]{#2}` – applies the transformation specified in `[#1]` on the file named in `#2` and delivers the value returned by that transformation. The first parameter of the user defined transformation function will receive the contents of the file as a Lua string. Since the data in the filevault are left intact, this macro will work transparently in either an active or inactive filevault.
The number of applications one can think of are manyfold. To name some: returning the length of the file, changing everything to uppercase, checking xml for correctness.
The argument `[#1]` must begin with the name of the Lua function performing the transformation. After a comma there may follow any number of comma separated arguments that are to be passed to the transformation function.[1]
To clarify this a complete example is presented of a function that will return the length of a file, either counted as bytes or counted as UTF8 characters. *Nota bene:* see how our Lua function is registered in the function registry of the filevault, because without this registration it will not be found. File *length.txt* contains

German ü and è in French.

```
\startluacode
-- Define our namespace as xmpl.
xmpl = xmpl or {}
-- Return the (utf8) length of a file.
xmpl.filelength = function (filedata, encoding)
  if encoding and encoding == "utf8" then
    return utf8.len(filedata)
  else
    return string.len(filedata)
  end
end
-- Register this function as "filelength".
hvdm.registerfunction
  ("filelength", xmpl.filelength)
\stopluacode
```

Input `[filelength]{length.txt}` to `\FileVaultReturnTransform` results in 28 bytes and `[filelength,utf8]{length.txt}` in 26 UTF8 characters.[2]

▷ `\FileVaultTransform[#1]{#2}` – nearly same as the previous macro but instead of leaving the data in the vault untouched, it will replace them by the result of the transformation. Because of this the macro is necessarily inoperative when the filevault is inactive and it will do so silently except for a

message in the log.
The return of this macro depends on the behaviour of the transformation function. Functions in Lua may return more than one value and this macro is programmed to accept one or two return values. The first value is used to replace the data in the filevault and the second value will be sent to the caller. In this manner it is possible to deliver a message, for instance to inform the caller of the cause that made an operation fail. Being more specific: an absent or nil second value returns nothing, otherwise that second value is returned. An example of its use might be the stripping of ignorable whitespace from xml: returning the stripped data twice will both change the data in the filevault as well as having it available for immediate processing.
Extending the previous example we will use this macro to replace the *length.txt* file by its length as UTF8 string. Note that the original file on disk will not be affected! Thus after `\FileVaultTransform[filelength,utf8]{length.txt}` calling `\FileVaultFileString{length.txt}` demonstrates that its value in the filevault has become

26

▷ `\FileVaultClear`
`\FileVaultClearFile[#1]` – respectively clears the FileVault completely or clears file `#1`; if that file is not in the vault nothing happens.
Example: after `\FileVaultClearFile[example.txt]` the vault contains:

length.txt *otherexample.txt*

and after `\FileVaultClear` we will receive an empty string "".

Information on Files
It sometimes is of interest to know if a file has a certain suffix. Perhaps in order to differentiate between files with the same basename but of different type: text, xml, images, pdf. Next follows macros used to query filenames and to retrieve the content of a directory.

▷ `\FileExist{#1}` – returns true or false on existence of the file `#1`.
Example: called on the source of this article naturally results in *true*.

▷ `\FileBaseName{#1}` – returns the filename stripped of prefixed directories.
Example: `./Documents/Letters/lastletter.doc` becomes *lastletter.doc*.

▷ `\FileDirectoryName{#1}` – returns the prefixed directories of the filename.
Example: `./Documents/Letters/lastletter.doc` becomes *./Documents/Letters/*.

▷ `\FileSuffix{#1}` – returns the suffix of the filename.
Example: `./Documents/Letters/lastletter.doc` returns *doc*.

▷ `\FileSuffixList{#1}{#2}` – returns the suffix of the filename if that suffix is in comma separated list `#2`.
Example: suffix list `{pdf,doc,txt}` returns *doc*, while `{pdf,txt}` will return an empty string.

▷ `\FileDirectoryList[#1]{#2}{#3}` – collects in a list all filenames in directory `#2` (empty is current directory) having suffix `#3` (empty suffix lists all files). The result is not directly returned but saved into a Lua table named `#1`. That list then can be queried by the table macros described below. With empty `#1` the name of the list is `FILEDIRECTORYLIST` by default. The information is added to the list if it already exists, except for the default list which is cleared before the operation.
Example: On this run there are *12 file(s)* in the current directory of which *5 tex-file(s)*.

Operations on Files
The purpose of the macros below is to execute file operations that otherwise would need intervening actions outside the ConTEXt run. The operations are concatenation of files, removal of files and querying for existence on the internet.

▷ `\FileConcatFile{#1}{#2}` – concatenates all files in directory `#1` having suffix `#2` and returns the concatenated contents. Before returning the intermediate temporary is removed.

▷ `\FileConcatName{#1}{#2}` – nearly the same as `\FileConcatFile` but here the temporary is not removed and its name is returned to the caller.

▷ `\FileConcatFilePrePost{#1}{#2}{#3}{#4}` – same as `\FileConcatFile` but `#3` precedes the concatenated contents while `#4` is affixed to the end.

▷ `\FileConcatNamePrePost{#1}{#2}{#3}{#4}` – same as `\FileConcatFilePrePost` but returning the filename instead of the content.

▷ `\FileRemove[#1]` – remove the file by name.

▷ `\URIReturnCode{#1}` – reaches into the internet with a `socket.http.request` to check if file given can be accessed. Returns the standard return code 200 on success and 404 for file not found. Waits 5 seconds for reaction from the internet before giving up.

Care is taken to change spaces in the URL to the mandatory %20.

Creation of Lua Lists
Collection of macros to produce and manipulate Lua tables. The tables or lists as named in the macros, are kept inside the module in an invisible table functioning as the holder of those created by the caller.
The macros in this section arose from the need to collect various information on the fly, storing it and present it afterwards in various forms.

Where lists are involved their name is always in `#1`.

List creation and removal

▷ `ListCreate[#1]` – creates a named list. That name is used in later accesses to the list. The list is created empty. Beware: reusing a list is silently inhibited, first delete an existing list before reusing the name.

▷ `\ListExist[#1]` – queries the existence of list `#1` and returns `true` or `false` accordingly.
Example: `\ListCreate[test]` then `\ListExist[test]` returns *true*.

▷ `\ListDelete[#1]` – removes the list by setting the reference to the list `nil`.
Example: given the above ListCreate the sequence `\ListExist[test]`, `\Delete{test}`, `\ListExist[test]` returns first *true* and then *false*.

▷ `\ListClear[#1]` – removes the content of the list, leaving it empty.

▷ `\ListCount[#1]` – returns the number of elements in the list. Lua tables have both an array and a key based section. The count is done over both these sections.
Example: empty list should return zero `\ListCreate[zero]` then `\ListCount[zero]` returns 0.

Addition of List Elements

▷ `\ListAdd[#1]{#2}` – adds `#2` as next element in the array section of the Lua table. Successive additions of the same element become successive elements in the list.
Example: add string "one" to the array section of list zero created above, the reported element count is 0 before and 1 after.

▷ `\ListAddKey[#1]{#2}{#3}` – adds element with key `#2` and value `#3` to the key section of the Lua table. Successive additions with the same key silently overwrite the previous value.
Example: add string "two" with key "second" to the same list with `\ListAddKey[zero]{second}{two}` and observe the incremented element count 2.

▷ `\ListAddSubKey[#1]{#2}{#3}{#4}` – adds element with key `#3` and value `#4` to a sublist keyed by `#2`, creating that sublist if necessary.

▷ `\ListAddTo[#1]{#2}` – adds element with key `#2` to the list setting its count to 1. Successive additions at the same key increment the counter value. The list therefore keeps a count of how often that key has been added.

▷ `\ListAddToKey[#1]{#2}{#3}` – adds to the array section of the Lua table a subtable `{#2,#3}` as a key-value pair.

Retrievial of List Elements

▷ `\ListArrayValue[#1]{#2}` – returns the element with index `#2`. If the index is outside the range of stored elements an empty string is returned.
Example: retrieve the first element in the array section of list zero `\ListArrayValue[zero]{1}` is *one*.

▷ `\ListKeyValue[#1]{#2}` – returns the value of the element at key `#2`. If the element is not present then an empty string is returned.
Example: retrieve the element with key "second" from list zero `\ListKeyValue[zero]{second}` is *two*.

▷ `\ListKeyValueWithDefault[#1]{#2}{#3}` – same as `\ListKeyValue` but instead of returning an empty string for an absent element, `#4` is returned as default instead.
Example: an element with key "third" has not been added thus `\ListKeyValue[zero]{third}` returns "", an empty string. The next call shows the return of a default `\ListKeyValueWithDefault[zero]{third}{three}` results in *three*.

▷ `\ListSubKeyValue[#1]{#2}{#3}` – returns element `#3` from sublist `#2`, empty string if absent.

▷ `\ListSubKeyValueWithDefault[#1]{#2}{#3}{#4}` – returns element `#3` from sublist `#2`, default `#4` if absent.

▷ `\ListValueKey[#1]{#2}` – find and return from the list the first key having `#2` as its value.

▷ `\ListValueSubKey[#1]{#2}{#3}` – find and return from the list the first key in sublist `#2` having `#3` as its value.

▷ `\ListValueSubKeyAll[#1]{#2}` – find and return from the list and all of its sublists the first key having `#2` as its value.

▷ `ListPrint[#1]` – simple printer for the contents of list `#1`.
The underlying Lua tables harbour two sections: (1) an array section indexed upwards from 1 (by default), and (2) a section with key-value pairs.

Both sections are printed unsorted, just as the entries are encountered by table traversal. Line endings are set to `\crlf` to accommodate ConTEXt. Example: a list has been made with two items in both the array and the key-value section.
`\ListPrint[Test]` prints:
1 = array item 2 added first
2 = array item 1 added last
1 = array item 2 added first
2 = array item 1 added last
secondkey = value two
firstkey = value one

XML-related operations

The purpose of next macros is the production of content to be handled by an XML processor. They provide for the contruction of (embedded) nodes, sorting lists of nodes, processing and checking of XML from various sources.

▷ `XMLContentToNode{#1}{#2}` – returns the string `<node>content</node>` where node = `#1` and content is `#2`. Convenient when a list must be filled with XML nodes to be processed later.

▷ `\ListToNodes[#1]{#2}`
`\ListToNodesSorted[#1]{#2}{#3}` – returns the content of the key section of Lua table `#1` as concatenated pairs <name><key>thekey</key><value>the-value</value>...</name> where `name` is `#1`, the name of the list. In the second macro the nodes are sorted to the keys with `#3` is `normal` for ascending (default) and `reverse` for descending keys.[3] This macro is meant for lists filled with `\ListAdd`.

▷ `\ListValuesToNodes[#1]{#2}`
`\ListValuesToNodesSorted[#1]{#2}{#3}` – same as `\ListToNodes` except here the sorting is for lists filled with `\ListAddKey` where the elements are tables containing a key-value pair.

▷ `\XMLProcessBuffer{#1}` – processes buffer `#2` containing a valid XML file with macro call `\xmlprocessbuffer{id}{#2}{}`. The `id` is filled by the called function.

▷ `\XMLProcessFile{#1}` – same as above with file `#1`. As a bonus the macro discerns the presence of embedded TEX and switches processing as described in the section on the FileVault.

▷ `\XMLProcessFolder{#1}` – same as above for all files in directory `#1` having the `xml` extension.

▷ `\XMLProcessString{#1}` – same as above with the content of argument `#1`.

▷ `\XMLEntitiesRead[#1]` – register entity declarations for XML processing from a dtd file `#1` with the ConTEXt procedure `xml.registerentity()`.

▷ `\XMLCheck[#1]{#2}` – checks the validity of the XML tree `#2` located in `#1` being `file`, `folder`, `buffer` or `string`. If the XML is correct an empty string is returned otherwise the string contains the nodes remaining after removal of the correct nodes. Example: `<root><node att="abc">error</root>` is missing the xml-header (not considered a problem) and a closing `</node>`. Checking results in
`>>> <root><node></root>`
which should be helpful in the repair. The check works by deleting correct nodes, starting within and working outwards. At the end of the reduction correct XML leaves nothing but an empty string, otherwise something is amiss as can be seen in the example.

Date and Time formatting

Dates can be given in the formats yyyy-mm-dd, yymmdd (20th century only), yyyymmdd, dd-mm-yyyy, d-m-yyyy, dd-m-yyyy, d-mm-yyyy. Negative dates or dates with BC get a negative year. Use yyyy for year only and yyyy-yyyy for a year range. Dates are checked for validity. Dates containing other characters than digits, dashes and possible BC are taken as is and considered dates already in final format. The formatters return the string `"DATE ERROR"` when something is amiss.

▷ `\DateCheck{#1}` – returns string `"true"` if a valid date has been found, `"false"` otherwise. A range of years yyyy-yyyy and a date already formatted return `"date"`.
Example: `\DateCheck{1-1-1900}` is *true* and for `13-13-1313` and `30-2-2020` it is *false* and *false*.

▷ `\DateFormat[options]{#1}` – formats the date in European format dd-mm-yyyy where days and months less than 10 are typeset with one digit only. There are both single and key=value options. The formatting options are `zero` (zero fill), `short` (abbreviated month name), `long` (full month name), `julian` (Julian date), `text` (no formatting), default is a compact format. A language option switches the translation (see the example below) default is the value of `\currentlanguage`.
Examples:
`\DateFormat[]{200201}` = *1-2-2020*
`\DateFormat[zero]{200201}` = *01-02-2020*
`\DateFormat[short]{200201}` = *1 Feb 2020*
`\DateFormat[long]{200201}` = *1 February 2020*
`\..[long,language=fr]{200201}` = *1 février 2020*
`\DateFormat[julian]{200201}` = *2458881*
`\DateFormat[text]{200201}` = *200201*

▷ `\DateCurrent` – the date of today *19-3-2021* formatted

with `\DateFormat`. Also one can obtain *19 March 2021* from `\DateFormat[long]{\DateCurrent}`.

▷ `\DateJulian{#1}` – same as `DateFormat[julian]{#1}`. The date converted to a Julian date with range limited from 4713 BC to 3628 AD. Today is *2459293* in Julian. The Julian date is useful when things have to be sorted on date.

▷ `\TimeFormatMinutes{#1}`
`\TimeFormatSeconds{#1}` — format time duration given as minutes or seconds respectively, into hh:mm:ss. Input with one or more :'s or otherwise not a number is considered formatted.
Examples:
`\TimeFormatMinutes{59}` = *00:59:00*
`\TimeFormatMinutes{120}` = *02:00:00*
`\TimeFormatSeconds{10}` = *00:00:10*
`\TimeFormatSeconds{3599}` = *00:59:59*
`\TimeFormatSeconds{3600}` = *01:00:00*

▷ `\TimeCurrent` – the time from the current clock as of the moment of typesetting is 10:04:00. Note that the underlying macro `\the\normaltime` returns the time in minutes since mdidnight.

Case Change and Character Selection

Changing case is notoriously difficult, at least in my experience. I always had trouble with `\uppercase` and friends. ConTEXt provides macros `\Word`, etc. but doing it yourself is a challenge I am not always able to resist. The implementation of CamelCase transformations came as a bonus.

▷ `\ChangeLower{#1}` – changes all letters in `#1` to lower case. For instance the french Était becomes était.

▷ `\ChangeUpper{#1}` – changes first letter to upper case, était thus becomes Était.

▷ `\ChangeUPPER{#1}` – changes all letters in `#1` to upper case, était becomes ÉTAIT.

▷ `\ChangeCamel{#1}` – changes all letters following whitespace in `#1` to upper case.
Example: *sample text word-combination* becomes: *Sample Text Word-combination*

▷ `\ChangeCamelPlus{#1}{#2}` – changes to uppercase in `#1` all letters following whitespace and those from `#2`. Beware: the - for example is special in Lua search patterns and therefore must be preceded by a %. Note that `\letterpercent` is how to insert it.
The same example with the - added:
Sample Text Word-Combination

▷ `\TypeCheck{#1}` returns the Lua type of `#1`. Useful to test if `#1` is a string that Lua can convert into a number.
Example: "5" has type *number* and "a5b" type *string*.

The macros below are intended to extract characters and truncate strings to substrings. For example in operating systems long filenames are sometimes truncated by removing parts in the middle.

▷ `\StringFirstCharacters{#1}{#2}` – truncates string `#1` to a length of `#2` by removing the excess characters at the end.
Example: string has more than 40 characters *This was a very long string more than th....*

▷ `\StringLastCharacters{#1}{#2}` – truncates string `#1` to a length of `#2` by removing the excess characters at the front.
Example: string has more than 40 characters *...ong string more than the available space.*

▷ `\StringFirstLastCharacters{#1}{#2}` – truncates string `#1` to a length of `#2` by removing the excess characters in the middle.
Example: string has more than 40 characters *This was a very lo...he available space.*

▷ `\CharacterSelect[#1]{#2}{#3}` – returns from string `#2` the `#3`-th character or an empty string if that character is not found. The option `#1` is a selector from `alphanum`, `alpha`, `digit`, `punct`. A recognized option returns the n-th character from the corresponding set, while an empty selector returns just the n-th character. The function does not know about UTF8 characters when a selector is given.
Example: `\CharacterSelect[]{the 7 dwarfs.}{7}` = d
Example: `\CharacterSelect[alphanum]{..}{7}` = a
Example: `\CharacterSelect[alpha]{..}{7}` = r
Example: `\CharacterSelect[digit]{..}{1}` = 7
Example: `\CharacterSelect[punct]{..}{1}` = .
Example: `\CharacterSelect[]{pygmée}{5}` = é

Numbers and Number Series

▷ `\RandomSeed{#1}` – initializes the Lua random generator with seed `#1`.

▷ `\RandomValue` – returns next random value from the Lua random generator.
Example: 0.85193537224893.

▷ `\RandomRange{#1}` – returns a random value within range 1-`#1` from the Lua random generator.

▷ `\Series[options]{#1}` – returns a series of `#1` numbers. The direction of the values can be ascending (default) or descending, the corresponding key options being `normal` (or empty) and `reverse`.

The start value and the stepsize follow from `start=number` and `step=number`, both having default 1. The blank separator between the list items can be changed with the option `separator=value` or `separator={value}`.[4] See the last example below where the default space separator is replaced by space + space.
Examples:
`\Series[]{10}` *1 2 3 4 5 6 7 8 9 10*
`\Series[reverse]{10}` *10 9 8 7 6 5 4 3 2 1*
`\Series[reverse,start=0]{10}` *9 8 7 6 5 4 3 2 1 0*
`\Series[start=0,step=-0.5]{4}` *0 -0.5 -1.0 -1.5*
`\Series[separator={ + }]{4}` *1 + 2 + 3 + 4*

A more elaborate example is the following:[5]

```
\leavevmode
\setupframed[extras=\space,width=5mm,height=5mm]
\ProcessCommaList{framed[framecolor=blue]}
    {\Series[separator={,}]{5}}}
```

1 2 3 4 5

A few remarks are in order. Macro `\ProcessCommaList` is a wrapper around `\processcommalist` which would otherwise crash as used here.[6]

▷ `\RandomSeries[#1]{#2}` – returns `#2` numbers randomly from the range 0 to 1. With option `#1` is `range=integer_number` the values are integers drawn from the interval 1..range. The option `#1` will receive an item separator as in the above example.
Example: `\RandomSeries[range=10]{4}` *4, 7, 2, 8*

▷ `\MDfive{#1}` – converts the string `#1` into MD5 hash value. Although nowadays not strong enough for a secure hash, it is sufficient to fingerprint (long) strings. Stored in a list useful to detect if these strings were encountered before.
Example: `\MDfive{[[ MD5 ]]}` is *6b3663a615846322674e3abf4fd59672*

▷ `\SafeNumber{#1}{#2}` – Return `#1` if the Lua function tonumber succeeds, otherwise return `#2` as default.
Example: strings 207 and 207a with default NAN, return respectively *207* and *NAN*.

Availability

The module and its supporting modules can be downloaded from my site `hvandermeer.com/publications.html`. The TEX-stuff resides in section "Articles on TeX", downloads are a little below in the link "ConTeXt module distribution".

Notes

1. Courtesy of the fact that Lua functions can accept a variable number of parameters.
2. For those who would expect 27 and 25 as answers: the newline at the end of the line is included in the count.
3. The tablesorter is derived from *Programming in Lua* by Roberto Ierusalimschy, 3rd edition, section 20.2 page 197.
4. The braces are mandatory in case certain characters are present in the option value, especially spaces, commas and Lua special pattern matching characters. Without the braces the underlying Lua function does not treat the option value as intended. The Lua special characters are `^$()%.[]*+-?`
5. The `\leavevmode` is needed to suppress the newlines that otherwise appear between the frames.
6. All parameters to the `\framed` could have been placed inside the `[]`'s, but it would clutter this presentation too much.

Hans van der Meer
havdmeer@ziggo.nl

GUST e-foundry font projects, closing report 2019–2020

For the record

The GUST e-foundry's set of interrelated projects that are reported on here was conceived in 2015. A leaflet presenting the ideas and asking for financial support was sent out to various TEX LUG boards later that year. Support was offered in 2015 by NTG, in 2016 by CSTUG and ConTEXt Group. DANTE e.V. and TUG joined in 2017.

The "advertising" leaflet mentioned above was turned into a one page summary and published in *TUGBoat*, Volume 38 (2017), No. 2 as "GUST e-foundry current font projects".

The official start of the project was never declared, but it seems that 2017 is a good number. However, work was being done already in 2016.

What was planned

The main goal of those projects was to add mathematical, technical and geometrical symbols to all of the TEX Gyre text fonts with the exception of TG Chorus. TG Chorus was excluded as such symbols seem of little use in a chancery font.

Further, several related ideas were coined:

- a sans-serif math OTF font, possibly based on DejaVu, for use in headings;
- a heavy math OTF font, possibly based on TG Termes, also for headings;
- a monospace text font with math symbols, for use in text editors.

Two other goals were also set:

- enhancements to existing math fonts, like math kerns, variant extra alphabets (e.g., calligraphic or double-struck) implemented using the "stylistic set" features `ss01`–`ss20`;
- continuous, yearly maintenance reviews and, if needed, releases of e-foundry's fonts with fixes.

Stage 1: what was done until 2019

The outcome of a part of the project that might be called its first stage was described in the paper by B. Jackowski, P. Pianowski, and P. Strzelczyk „TEX Gyre text fonts revisited", published both in *TUGBoat*, Volume 39 (2018), No. 3 and Die Technische Komödie, 30. Jahrgang, Heft 3/2018.

This is a crude summary of what was done (for details see the article):

- devising the enhanced repertoire of glyphs;
- elements of MetaType 1 (`en.wikipedia.org/wiki/METATYPE1`) were reimplemented by replacing T1utils and some AWK and Perl scripts with Python code interfacing to FontForge – both more portable and easier to maintain;
- the internal structure of the TG fonts became even more OTF-like:
  - the `ss10` feature allows the use of the original math symbols if replacements are not liked or needed and
  - the "anchors" mechanism based on the `ccmp`, `mark` and `mkmk` features is used to place accents over glyphs in a precise way;
- the improved MetaType 1 was used to extend the list of glyphs of TG Adventor and TG Pagella by over 850 items, which took the fonts to ver. 2.501

Stage 2: Algotype, the successor to MetaType 1, 2019–2020

After releasing the new versions of TG Adventor and Pagella, the team decided to attempt a full hearted reimplementation of MetaType 1.

It is important to notice that up to now for over 20 years all of the many e-foundry's fonts were produced with MetaType 1. It began in late nineties of the twentieth century with a no-name engine to create Adobe PostScript Type 1 outline fonts for Janusz M. Nowacki's efforts to revive the traditional Polish type Antykwa Półtawskiego and was reported at the Heidelberg EuroTEX Conference in 1999 ("Antykwa Półtawskiego: a parameterized outline font").

A few years later OpenType became ISO standard (Open Font Format: ISO Standard ISO/IEC 14496-22 [MPEG-4 Part 22], March 2007). Obviously, the engine by the name MetaType1 has been adapted and OpenType versions could be included in the TEX Gyre collection of fonts (released in 2006–2007).

Another adaptation of MetaType 1 became necessary with the advent OpenType Math font when in 2010 Microsoft implemented math fonts support into MS Office. MetaType 1 proved itself by generating the TG Math fonts: Bonum, Pagella, Schola, Termes and later DejaVu. The engine was also used by the e-foundry team for Latin Modern fonts in both Type 1 and OpenType formats along with the LM OpenType math font.

All those changes accumulated over so many years lead inevitably to MetaType 1 being unwieldy and complex. In particular, porting of the system became a nightmare, which was experienced when Marek Ryćko had to step in for Piotr Strzelczyk who left the team in early 2019 and MetaType 1 had to be installed from scratch in a different environment.

Leaving by Piotr Strzelczyk was a severe blow and was bound to a drastical change in priorities: nothing became more important than a reimplementation and redesign of the font production line. At BachoTEX 2019 "Redesign of a Metapost-based font generating system" by Marek Ryćko and Bogusław Jackowski, presented by Marek Ryćko was awarded the W. J. Martin Prize.

MetaType 1 was rewritten in such a way that only MetaPost and Python 3 (with some pieces of Python 2 to communicate with the FontForge library) are used. Moreover, a new way of configuring of the system was worked out – the configuration is now governed by simple, universal data files (in JSON format). Exactly the same scripts can be run both under Linux and Windows (no tests with Macintosh were performed so far) which solved the portability problem.

The new engine is called Algotype. The name tries to stress that fonts are being defined algorithmically. The Python part of Algotype is now available at `pypi.org`.

The team is going to publish the Algotype system on GitHub.

Current and future font works, 2021–...

Immediate future:[1]

- Despite a lot of effort already devoted to Algotype, it still does require work. Nonetheless it is productive – it was developed and tested doing real work. Enhanced (see: "What was planned") TG text fonts Schola and Termes are close to being released together with revised versions 2.501 of TG Adventor and TG Pagella.
- There is hope for a new release of the Latin Modern fonts with corrections proposed by Frank Mittelbach at BachoTEX 2019 to be in time for the 2021 release of TEX Live.
- 2021 should see the rest of the enhancements to the TEX Gyre family, i.e., the new releases of TG Bonum, TG Cursor and TG Heros.

The renewed team with Marek Ryćko hopes to be able to tackle in the near future the remaining tasks listed in section "What was planned" although prefers not to make too many promises.

Financing (support) up to date

The following donations to the project were received and paid out up-to-date:

- ConTEXt Group: 1,500 EUR in the years 2017–2019;
- $\mathcal{CS}$TUG: 2,000 EUR in the years 2017–2018;
- DANTE e.V.: 7,000 EUR in 2018;
- NTG: 18,000 EUR in the years 2015–2020;
- TUG: 2,903 USD in 2017;
- individual persons: 1,960 PLN in the years 2017–2019.

The total funding amounted to 28,500 EUR, 2,903 USD and 1,960 PLN.

The GUST e-foundry is really very, very grateful to its supporters and promises to continue its best efforts.

Final remarks: feedback craved for

The gentle readers of this report are kindly asked for feedback: do you like/hate/see faults in/ask for enhancements to/propose fixes to/ ... the works of the GUST e-foundry?

Please write! The e-foundry will do its best to satisfy your request.

Notes

1. It should be noted that the first two items would have had already happened if it were not for the COVID-19 pandemic and Bogusław Jackowski being hospitalized for over a month for a COVID-19 infection and a heart surgery.

Jerzy Ludwichowski
GUST, Toruń, Poland
Jerzy.Ludwichowski (@) gust.org.pl

De ontwikkeling van het LaTeX package `fancyhdr`

Een historisch en technisch overzicht

Abstract
Dit artikel geeft een overzicht van de ontwikkeling van het LaTeX package `fancyhdr`, en de hulpmiddelen die ik hiervoor gebruik. Ook wordt een overzicht gegeven van de manier van testen.

Keywords
LaTeX, package, fancyhdr, headers, footers, versiebeheer, testen

Inleiding

Dit artikel beschrijft kort de technieken die ik gebruik bij het (verder) ontwikkelen van het LaTeX package `fancyhdr`. We beginnen met een overzicht van hoe dit package ontstaan is. Daarna beschrijf ik de wensen die ontstonden voor een nieuwe versie. De ontwikkelingen noodzaakten om het beheer van verschillende versies goed ter hand te nemen. Daarom beschrijf ik de verschillende systemen voor versiebeheer, en hoe ik die toepas in de ontwikkeling. Tenslotte beschrijf ik de manier van testen, wanneer nieuwe features aan het package toegevoegd worden of wanneer problemen opgelost worden. In het bijzonder wordt uitgelegd hoe het testen grotendeels geautomatiseerd kan gebeuren.

Geschiedenis

Ik kan me niet meer herinneren wanneer ik met het ontwikkelen van het package `fancyhdr` begonnen ben. Het moet in het begin van mijn escapades met LaTeX zijn geweest. Ik heb er geen dagboek van bijgehouden. Informatie over de oudste versie die ik nog heb teruggevonden was die van versie 1.4 van 16 september 1994. Deze versie zelf heb ik niet meer, maar in oudere versies is dit de oudste versie die vermeld wordt.

Wat ik me herinner is dat in de begin-dagen van LaTeX er niet veel mogelijkheden waren om de headers en footers van LaTeX-documenten aan te passen. Er waren een paar packages die dat deden, o.a. een package met een mogelijkheid om “three-part header” te maken, d.w.z. een header die bestond uit een linker-, midden- en rechterdeel (idem voor footers), en een ander package dat de mogelijkheid gaf om een streep onder de header te zetten. Helaas konden deze beide niet gecombineerd worden, terwijl dat toch een populaire keuze is. Daar wilde ik verandering in brengen door deze beide faciliteiten te combineren in één package. De naam werd ‘fancyheadings’.

In het begin was dit een simpel package, maar in de loop van de jaren werd het uitgebreid, en bovendien robuuster gemaakt, omdat er altijd weer situaties worden ontdekt waarbij interactie met andere packages, of met bepaalde LaTeX-constructies, ongewenste effecten geven. Voor de huidige faciliteiten die het package biedt, zie de documentatie (`texdoc fancyhdr`).

Zoals je kunt zien is de naam van het package onderweg veranderd van 'fancyheadings' naar 'fancyhdr'. De redenen hiertoe geven een interessant kijkje in de informatica-archeologie. Ik heb het package ontwikkeld op Unix-systemen, die destijds op de universiteit de werkomgeving bepaalden. Echter, de meeste gebruikers hadden indertijd MS-DOS (ja, zover gaan we terug in de geschiedenis). En MS-DOS gebruikte bestandsnamen van 8.3, dat wil zeggen 8 tekens voor de punt en 3 erna. Grotere namen werden wel geaccepteerd (althans vóór de punt), maar wat overtollig was werd gewoon weggegooid. Dus het package werd op MS-DOS opgeslagen als `fancyhea.sty`. Dit bracht sommige lieden ertoe om zich wat typewerk te besparen door 'fancyhea' als package naam te gebruiken. Dit was ook nog in de tijd voor LaTeX 2_ε, zodat het gespecificeerd werd als

```
\documentstyle[fancyhea]{article}
```

in plaats van

```
\documentstyle[fancyheadings]{article}
```

Alle packages (*styles* genoemd) moesten in die header gespecificeerd worden, dus het is enigszins begrijpelijk dat geprobeerd werd om dat zo compact mogelijk te houden. Het probleem was echter, dat het fout ging wanneer zo'n document op een Unix-systeem verwerkt werd omdat daar niet een '`fancyhea.sty`' aanwezig was. Daarom heb ik op een gegeven moment besloten om het package een nieuwe naam te geven die voldeed aan de 8.3 beperking, namelijk `fancyhdr.sty`. Overigens bevat de distributie nog steeds een `fancyheadings.sty`, die je waarschuwt om niet meer 'fancyheadings', maar 'fancyhdr' te gebruiken.

Versie "management"

Het beheren en uitbrengen van nieuwe versies was in het begin simpel. Er was een bestand `fancyheadings.sty` en later dus `fancyhdr.sty` met de code en een bestand `README` of `README.txt`, waarin de commando's beschreven waren. Het `.sty` bestand bevatte de code met ertussendoor commentaar achter `%`-tekens. Vooraan werd een overzicht gegeven van de versies, te beginnen met versie 1.4 zoals hierboven vermeld. Iedere nieuwe versie kreeg een alinea aan het eind met het versienummer, de datum en een korte beschrijving van de wijzigingen. Voor een voorbeeld, zie Appendix A.

Documentatie door George Grätzer

Zoals hierboven vermeld bestond de eerste 'documentatie' uit een `README` documentje met de beschrijving van de commando's van het package, zonder verdere uitleg erbij. Het begin van de echte documentatie werd gevormd door een artikel van George Grätzer (auteur van het boek *Math into LaTeX*[1]) in de Notices van de American Mathematical Society[2]. Hij bood mij dit artikel ook aan als documentatie voor het package. Ik heb het stuk uitgebreid en aangepast, maar er zijn nog steeds stukken in de huidige documentatie die meer of minder lijken op dat oorspronkelijke artikel.

Sinds LaTeX 2_ε is de standaard werkwijze voor packages, dat de documentatie en de code samen in een `.dtx` bestand worden gezet. Dit levert een vorm van *Literate Programming* op zoals gepromoot door Donald Knuth. Ik heb lang gewacht om de code plus documentatie om te zetten, omdat het systeem van een aparte documentatie met de code in een `fancyhdr.sty` bestand prima voldeed, en het best een werk was om het om te zetten. Uiteindelijk ben ik op 11 oktober 2016 begonnen met het omzetten. Dat was niet meer dan `fancyhdr.sty` versie 3.8, ingepakt in `fancyhdr.dtx`, zonder de documentatie. Enkele dagen later heb ik ook de documentatie toegevoegd, en het additionele package `extramarks`, en het oude `fancyheadings`. Maar pas op 25 januari 2019 werd de hele distributie gebaseerd op `fancyhdr.dtx` (versie 3.10).

Wensen voor versie 4

Aan het einde van versie 3 van `fancyhdr` was het duidelijk dat er een aantal ontwerpbeslissingen in zaten die suboptimaal waren. Zo controleert fancyhdr o.a. of er genoeg verticale ruimte gereserveerd is voor de headers en footers. Zo niet dan wordt er een waarschuwing gegeven, en in versie 3 werd dan ook die ruimte aangepast voor de volgende pagina's (in LaTeX resp. `\headheight` en `\footskip`), zodat er daarna geen meldingen meer werden gegeven. Helaas zorgde dit regelmatig voor onaangename verrassingen, omdat de paginalayout dan niet meer consistent is.

Een ander gebrek was dat de definities die de headers en footers bepalen, globaal werden gedaan, dus niet beperkt tot de huidige scope (TeX group). Bij het wisselen van page styles in de loop van het document gaf dat vaak ongewenste effecten. Daarom liep ik al geruime tijd (enige jaren) rond met het idee om een echt verbeterde versie te maken, dus meer dan wat kleine aanpassingen. Alleen wist ik niet precies hoe dit aan te passen en eerlijk gezegd vond ik het ook weer te weinig dringend om daar veel energie in te stoppen. Maar het bleef knagen.

Mail van Frank Mittelbach

Op 15 november 2018 kreeg ik een e-mail van Frank Mittelbach, waarin hij vertelde dat hij bezig was met een nieuwe editie van *The LaTeX Companion*. Hij maakte mij attent op een package `nccfancyhdr` van Alexander I. Rozhenko, dat een aantal gebreken van `fancyhdr` opgelost zou hebben, en vroeg mij of ik van plan was dat in `fancyhdr` over te nemen.

Ik kende dat package niet, maar besloot om het te bestuderen. Mijn conclusie was dat een aantal aspecten hiervan al opgelost waren, en dat er interessante ideeën in zaten die ik goed kon gebruiken. Ik heb toen een lijst gemaakt met wat ik in versie 4 zou gaan doen, en die heb ik aan Frank gestuurd. Zie de lijst in Appendix B.

Samen met mijn eerdere gedachten heb ik dit gebruikt voor het ontwikkelen van `fancyhdr` versie 4. Ook wilde ik de documentatie flink onder handen nemen. Op 15 maart 2019 begon ik met de ontwikkeling van versie 4.0beta.

Vanaf dat moment waren er enkele parallel lopende lijnen in de ontwikkeling:

- Ontwikkeling van nieuwe features voor versie 4.0
- Verbetering van de documentatie
- Testen van alle voorbeelden en maken van een test suite

Tijdens de eerste Corona-periode in de zomer van 2020 lag het werk grotendeels stil. Maar in december besloot ik het weer op te pakken. De code voor versie 4 was bijna klaar, maar de documentatie zou nog veel werk kosten. Ik heb toen besloten om de code af te maken, en te zorgen dat de documentatie voldoende was voor de nieuwe versie. Deze werd op 2 januari 2021 vrijgegeven op CTAN.

Het werk ging daarna door. De documentatie moest nog steeds afgemaakt worden, en een aantal voorbeelden uit deze documentatie waren nog niet gecontroleerd. Sommige moesten echt verbeterd worden, omdat ze net niet klopten.

Intussen kwamen er toch nog nieuwe tekortkomingen aan het licht, in het bijzonder door discussies en vragen op `tex.stackexchange.com`. Daarom heb ik toch weer enkele nieuwe functies toegevoegd, waarvan er een aantal al wel geïmplementeerd zijn, maar nog niet voldoende gedocumenteerd. Met andere woorden, versie 4.1 is nu in zicht. Intussen is er ook nog een versie 4.0.1 uitgebracht met een deel van de verbeteringen in de documentatie.

Versiebeheer

Bij de initiële ontwikkeling waren er twee praktische problemen die met versiebeheer te maken hebben.

1. Hoe houd ik bij welke versies er geweest zijn en wat de verschillen tussen de verschillende versies zijn?

2. Naarmate het project groter wordt, en ik aan verschillende aspecten (documentatie, bug fixes, nieuwe ontwikkelingen) aan het werk ben: hoe houd ik de verschillende lijnen uit elkaar? Bijvoorbeeld als ik met een nieuwe ontwikkeling bezig ben, die nog maar half voltooid is, en er moet plotseling een bugfix plaatsvinden op de oude versie.

Hierbij komen versiebeheersystemen (*version control/management systems*) te hulp. Dit zijn systemen waarin de geschiedenis van een project bijgehouden wordt, en waarin ook verschillende lijnen uitgezet, en later weer samengevoegd kunnen worden. Deze systemen geven ook de mogelijkheid om met verschillende personen samen te werken aan een project, hoewel dat in mijn geval niet relevant is.

Initiële systemen: RCS en CVS

Er is een lange ontwikkeling geweest van versiebeheersystemen, zowel gratis beschikbaar (open source) als commerciële systemen. Ik beperk me hierbij tot de gratis systemen. De meeste van deze systemen zijn oorspronkelijk ontwikkeld op Unix(-achtige) systemen en daarna geporteerd naar bijvoorbeeld Windows en MacOS.

RCS was een simpel systeem waarbij je van een aantal bestanden versies kon bijhouden. Deze versies werden in een aparte map bijgehouden, en je kon een nieuwe versie van een bestand erin stoppen (*checkin*), of de laatste of een eerdere versie eruit halen (*checkout*). Bij de *checkin* geef je aan wat de reden ervan is (bijvoorbeeld een nieuw stuk code, een bugfix) en het systeem houdt ook bij wie dit gedaan heeft. Als iemand een *checkout* van een bestand doet met de bedoeling om wijzigingen aan te brengen, dan wordt er een *lock* op dat bestand gezet, zodat een ander er niet bij kan, althans niet met de mogelijkheid om te wijzigen. Dit om te voorkomen dat verschillende wijzigingen elkaar in de weg zitten of elimineren. Dit maakt het samenwerken lastiger, want je moet dan afspreken wie er aan de beurt is om aan een bepaald bestand te werken.

Een ander nadeel van RCS was dat elk bestand een onafhankelijke eenheid was. Als je een wijziging uitvoerde waarbij twee of meer bestanden betrokken waren, dan hield het systeem niet bij dat deze een eenheid vormden. Dus als je naar een vorige toestand terug wilde moest je zelf uitzoeken welke versies van de bestanden bij elkaar hoorden. De versies waren genummerd maar het ene bestand kon meer wijzigingen gehad hebben dan het andere, en daardoor ook andere nummers.

Een verbetering was het systeem CVS, dat bovenop RCS gebouwd was. Het gebruikte dus wel dezelfde bestanden om de versies op te slaan, maar voegde daar een administratieve laag aan toe. De twee belangrijkste aspecten hiervan waren:

1. Je kon meer dan één bestand tegelijk inchecken, en die vormden dan samen een nieuwe versie.

2. De uitgecheckte bestanden werden niet meer gelockt. Meerdere personen konden dezelfde bestanden bewerken. Maar bij het inchecken werd er gecontroleerd of er geen conflict ontstond. Als dat het geval was, ging de *checkin* niet door en moest de ontwikkelaar uitzoeken hoe het conflict opgelost kon worden. Vaak was daar natuurlijk overleg tussen de betrokken personen voor nodig. Maar als verschillende mensen aan onafhankelijke delen van een project werken, is er vaak niets aan de hand, zelfs als ze aan dezelfde bestanden werken. Hierdoor wordt het werken efficiënter. Dit proces van het samenbrengen van verschillende ontwikkelingslijnen heet *merging*.

3. RCS bestanden moesten altijd locaal op een computer benaderd worden, maar CVS had ook de mogelijkheid om op een server te draaien, en kon dan vanaf andere computers benaderd worden (*client-server systeem*). Voor gebruik door één persoon werd het echter meestal in lokale bestanden gezet.

Een collectie van de versies van bij elkaar horende bestanden heet een *repository*. Het is mogelijk om verschillende *repositories* te hebben. Vaak werd er een *repository* per project gemaakt.

CVS biedt ook de mogelijkheid om expliciet verschillende onafhankelijke lijnen van ontwikkeling op te zetten. Dit worden dan *branches* genoemd. Ook hierbij kan op een gegeven moment *merging* gebruikt worden om deze samen te voegen.

CVS werd in 1986 ontwikkeld door Dick Grune aan de Vrije Universiteit in Amsterdam. Voor meer informatie over RCS en CVS zie `http://linuxdocs.org/HOWTOs/CVS-RCS-HOWTO.html`

Subversion (SVN)

Subversion is een versiebeheersysteem dat ontwikkeld is om een aantal tekortkomingen van CVS op te lossen. CVS bestond uit een verzameling scripts om het RCS systeem heen; SVN daarentegen is van de grond af aan ontwikkeld. Maar de principes zijn wel vergelijkbaar. Tegenwoordig is SVN een Apache project.

Een aantal belangrijke verschillen tussen CVS en SVN:

- CVS gebruikte RCS bestanden om de versies op te slaan; SVN gebruikt een eigen database systeem. Dit geeft de mogelijkheid om meer informatie op te slaan. Zo kan een gebruiker *attributen* aan een bestand koppelen.
- SVN is sneller, omdat het geoptimaliseerd geprogrammeerd is.
- CVS was bedoeld voor tekstuele bestanden (ASCII), terwijl SVN goed werkt met alle soorten bestanden.

Omdat SVN veel efficiënter en beter gestructureerd was dan CVS, heeft het vrij snel de functie van CVS overgenomen. Lange tijd is het het populairste versiebeheersysteem geweest, vooral in de Unix en Open Source wereld. Het wordt nog steeds gebruikt, omdat het relatief simpel is om ermee te werken. Maar het is intussen ook al weer ruim 20 jaar oud.

Gedistribueerd versiebeheer

Zowel CVS als SVN gebruiken een centraal *repository* om de versies op te slaan. Het repository is via een *client-server systeem* te benaderen door de gebruikers. Dit heeft het voordeel dat je op afstand met elkaar kunt samenwerken. Alle versie-informatie van een project is dus op één locatie aanwezig. Dit geeft een paar nadelen:

- Als er een storing is in de server, of in de internetverbinding van de server of de client, dan kan er geen nieuwe versie-informatie gemaakt worden. Een ontwikkelaar die al een kopie heeft van de *repositiory* (zo'n kopie wordt *workspace* genoemd) kan hieraan werken, maar als hij/zij een nieuwe versie wil inchecken, of een nieuwe *branch* wil beginnen, kan dat op dat moment niet.
- Alle nieuwe versies en branches moeten in de centrale *repository* opgeslagen worden en zijn dus ook voor iedereen zichtbaar. Lang niet alles is voor iedereen relevant; zo kan een ontwikkelaar even iets nieuws uitproberen, en hiervoor een *branch* aanmaken. Dit vervuilt het *repository* al gauw met informatie die alleen voor één persoon relevant is, en bij een groot project leidt dat onvermijdelijk tot een chaotisch geheel. En bovendien kan het leiden tot ongewenste conflicten in de versies. Om dit te vermijden heeft men dan de neiging om daarvoor maar geen nieuwe versies of *branches* aan te maken, maar dat vermindert nu weer het nut van het versiebeheersysteem.

Om deze bezwaren te overwinnen zijn de zogenaamde *gedistribueerde versiebeheersystemen* ontwikkeld. Hierbij heeft een project één (of meer) centrale *repositories* waar de informatie (versies en *branches*) in staat die voor het hele project van belang is. Daarnaast heeft iedere gebruiker een eigen *repository* waar hij zijn lokale versies en *branches* in opslaat. Hij kan informatie uit de centrale *repository* naar zijn eigen repository trekken (*pull* operatie) en omgekeerd zijn nieuwe informatie naar het centrale *repository* uploaden (*push* operatie). Dit kan per branch apart gedaan worden. Bij een *push* operatie kunnen er natuurlijk weer conflicten optreden waarbij een *merge* moet plaatsvinden.

Er zijn diverse van deze systemen o.a. Mercurial (hg) en Bazaar (bzr). Dit zijn vrij simpele systemen, gemakkelijk te gebruiken, maar ze zijn hun populariteit aan het verliezen. Omdat ze niet teveel ingewikkelde functies hebben maar zich richten op de basale functies *checkin, checkout* en het beheer van *branches* e.d. hebben ze voordelen voor mensen die niet teveel toeters en bellen willen. Ze hebben echter weinig nieuwe ontwikkeling. Ze zijn beide geschreven in Python versie 2, en deze versie is gestopt met ontwikkeling. Verschillende Linux systemen komen bijvoorbeeld niet meer met Python 2, maar Python3, wat niet compatibel is met Python 2. Alleen van Bazaar is er een nieuwe kloon ontwikkeld die Breezy heet en wel op Python3 is gebaseerd.

Het meest gebruikte gedistribueerde versiebeheersystemen is tegenwoordig Git. Het is ontwikkeld voor het beheer van de code van Linux, en wordt daarom door veel mensen gebruikt. Het heeft veel mogelijkheden die Bazaar en Mercurial niet hebben (net zomin als SVN) waardoor het voor fulltime ontwikkelaars in grote projecten erg interessant is, maar daardoor ook moeilijker te gebruiken.

Er zijn een aantal services waar Git *repositories* kunnen worden opgeslagen, zoals Github (`github.com`) en Gitlab (`gitlab.com`) waar je gratis open source (of andeszins publieke) *repositories* kunt aanmaken, en tegen betaling *repositories* voor particuliere projecten. Deze zijn zeer populair, o.a. de LaTeX groep heeft hier de repositories van LaTeX 2_ε[3] en LaTeX3[4].

Github en Gitlab hebben een aantal extra faciliteiten, o.a. een web-interface, waarmee je door de *repository* kunt browsen, bugs kunt rapporteren en verschillende versies van een project kunt downloaden als een zip-bestand.

Mijn huidige setup

Voor de ontwikkeling van mijn software en soms ook van gewone tekstuele of TeX-documenten gebruik ik tegenwoordig Git, vooral omdat dit uitwisseling met anderen gemakkelijker maakt, en ik de installatie ervan makkelijker vind dan die van Mercurial en Bazaar.

Ik heb o.a. *repositories* voor de LaTeX packages `fancyhdr` en `multirow`. Deze hebben ook een *repository* op Github, waar in ieder geval de (redelijk) stabiele versies resp. *branches* in zitten[5].

Voor `fancyhdr` heb ik op dit moment de branches

V3.10 de laatste release van `fancyhdr` versie 3

V4.0.1 op dit moment de laatste versie die vrijgegeven is

master de hoofd*branch*, meestal gelijk aan de laatste release.

V4.1beta werk aan de code van versie 4.1

new-documentation werk aan de documentatie

extramarks2 werk aan een nieuw package `extramarks2` (update van het `extramarks` package), wat een onderdeel van de `fancyhdr` distributie is

Het is de bedoeling dat de laatste 3 *branches* op den duur gemerged worden tot versie 4.1. Maar dit zal nog wel enige maanden duren. En verder gebruik ik af en toe nieuwe

branches voor wat experimenteel werk maar die verdwijnen meestal na een tijdje, òf omdat ze niet meer nodig zijn, òf omdat ze geïntegreerd worden in een andere *branch*.

De *branches* `master`, `V3.10`, `V4.0.1` en `extramarks2` zijn ook op Github aanwezig, en `V4.1beta` in feite ook omdat die een onderdeel is van de *branch* `extramarks2`. Maar deze *branch* is tijdelijk.

Examples

Tijdens het ontwikkelen van `fancyhdr` versie 4, en het aanpassen van de documentatie ben ik begonnen met alle voorbeelden in de documentatie te testen. Daarvoor heb ik van elk voorbeeld een LaTeX-document gemaakt gebaseerd op dit voorbeeld, om te kijken of de code werkelijk de layout produceert die de documentatie eraan toeschrijft. Dat bleek niet altijd het geval te zijn, wat in de meeste gevallen leidde tot het aanpassen van de code, of in een enkel geval aanpassen van de beschrijving. Dit proces is nog niet helemaal voltooid, ik ben ergens aangekomen bij "Example 34".

Daarnaast heb ik nog een grote verzameling testprogramma's die soms ontwikkeld zijn om nieuwe features te testen, en soms gebaseerd zijn op bug reports. Maar ook heel vaak zijn ze gebaseerd op vragen van gebruikers, vroeger vooral van de nieuwsgroep `comp.text.tex` en de mailing lijst `TEX-NL`, en tegenwoordig vaak van het forum `tex.stackexchange.com`. Deze heb ik de laatste twee jaar omgewerkt door ze vooraf te laten gaan door een inleiding die beschrijft wat het probleem is, en meestal ook de code noemt die gebruikt is (die natuurlijk ook in het document staat, maar niet noodzakelijk als tekst). Al deze LaTeX-documenten staan in een apart *repository* op Github[6]. Dit zijn nog niet alle testbestanden die ik heb; er zijn er meer die nog in die vorm met beschrijving gegoten moeten worden.

In figuur 1 staan een aantal voorbeelden van iets geavanceerder gebruik van `fancyhdr`. Deze voorbeelden zijn uit bovengenoemde verzameling gelicht maar zijn aangepast om in dit artikel te passen. Hier volgt een summiere beschrijving met de belangrijkste LaTeX-code van elk voorbeeld. In alle voorbeelden is een groter lettertype gebruikt in de header (en soms in de footer) om het leesbaarder te maken in de verkleinde versie.

figuur 1a Hierbij wordt een header gebruikt tussen twee horizontale lijnen. De onderste lijn is de standaard lijn die `fancyhdr` geeft (maar dan dikker). De bovenste lijn is een duplicaat ervan, expliciet ingevoegd met het commando `\headrule`.

```
\usepackage{fancyhdr}
\pagestyle{fancy}
\renewcommand{\headrulewidth}{2pt}
\fancyhf{}
\fancyhead[C]{%
  \headrule
  \vspace{12pt}
  {\LARGE Project Description}
  \vspace{8pt}
}
\fancyfoot[C]{\thepage}
```

figuur 1b In dit voorbeeld wordt de lijn onder de header vervangen door een wat decoratievere met behulp van een ornament uit het `fourier-orns` font. Boven de footer wordt ook zo'n lijn toegevoegd. Dit gebeurt door de commando's `\headrule` en `\footrule` te herdefiniëren.

```
\usepackage{fourier-orns}
\usepackage{fancyhdr}
\pagestyle{fancy}
```

a: header tussen lijnen

Project Description

1 Introduction

As any dedicated reader can clearly see, the Ideal of practical reason is a representation of, as far as I know, the things in themselves; as I have shown elsewhere, the phenomena should only be used as a canon for our understanding. The paralogisms of practical reason are what first give rise to the architectonic of practical reason. As will easily be shown in the next section, reason would thereby be made to contradict, in view of these considerations, the Ideal of practical reason, yet the manifold depends on the phenomena. Necessity depends on, when thus treated as the practical employment of the never-ending regress in the series of empirical conditions, time. Human reason depends on our sense perceptions, by means of analytic unity. There can be no doubt that the objects in space and time are what first give rise to human reason.

1.1 The Problem

Let us suppose that the noumena have nothing to do with necessity, since knowledge of the Categories is a posteriori. Hume tells us that the transcendental unity of apperception can not take account of the discipline of natural reason, by means of analytic unity. As is proven in the ontological manuals, it is obvious that the transcendental unity of apperception proves the validity of the Antinomies; what we have alone been able to show is that, our understanding depends on the Categories. It remains a mystery why the Ideal stands in need of reason. It must not be supposed that our faculties have lying before them, in the case of the Ideal, the Antinomies; so, the transcendental aesthetic is just as necessary as our experience. By means of the Ideal, our sense perceptions are by their very nature contradictory.

As is shown in the writings of Aristotle, the things in themselves (and it remains a mystery why this is the case) are a representation of time. Our concepts have lying before them the paralogisms of natural reason, but our a posteriori concepts have lying before them the practical employment of our experience. Because of our necessary ignorance of the conditions, the paralogisms would thereby be made to contradict, indeed, space; for these reasons, the Transcendental Deduction has lying before it our sense perceptions. (Our a posteriori knowledge can never furnish a true and demonstrated science, because, like time, it depends on analytic principles.) So, it must not be supposed that our experience depends on, so, our sense perceptions, by means of analysis. Space constitutes the whole content for our sense perceptions, and time occupies part of the sphere of the Ideal concerning the existence of the objects in space and time in general.

b: gebruik ornament

1 Introduction

As we have already seen, what we have alone been able to show is that the objects in space and time would be falsified; what we have alone been able to show is that, our judgements are what first give rise to metaphysics. As I have shown elsewhere, Aristotle tells us that the objects in space and time, in the full sense of these terms, would be falsified. Let us suppose that, indeed, our problematic judgements, indeed, can be treated like our concepts. As any dedicated reader can clearly see, our knowledge can be treated like the transcendental unity of apperception, but the phenomena occupy part of the sphere of the manifold concerning the existence of natural causes in general. Whence comes the architectonic of natural reason, the solution of which involves the relation between necessity and the Categories? Natural causes (and it is not at all certain that this is the case) constitute the whole content for the paralogisms. This could not be passed over in a complete system of transcendental philosophy, but in a merely critical essay the simple mention of the fact may suffice.

1.1 The Problem

Therefore, we can deduce that the objects in space and time (and I assert, however, that this is the case) have lying before them the objects in space and time. Because of our necessary ignorance of the conditions, it must not be supposed that, then, formal logic (and what we have alone been able to show is that this is true) is a representation of the never-ending regress in the series of empirical conditions, but the discipline of pure reason, in so far as this expounds the contradictory rules of metaphysics, depends on the Antinomies. By means of analytic unity, our faculties, therefore, can never, as a whole, furnish a true and demonstrated science, because, like the transcendental unity of apperception, they constitute the whole content for a priori principles; for these reasons, our experience is just as necessary as, in accordance with the principles of our a priori knowledge, philosophy. The objects in space and time abstract from all content of knowledge. Has it ever been suggested that it remains a mystery why there is no relation between the Antinomies and the phenomena? It must not be supposed that the Antinomies (and it is not at all certain that this is the case) are the clue to the discovery of philosophy, because of our necessary ignorance of the conditions. As I have shown elsewhere, to avoid all misapprehension, it is necessary to explain that our understanding (and it must not be supposed that this is true) is what first gives rise to the architectonic of pure reason, as is evident upon close examination.

c: woordenboekstijl

Dit voorbeeld gebruikt headers zoals in een woordenboek. Ieder lemma (item) gebruikt `\markboth{#1}{#1}` en de headers gebruiken `\rightmark` voor het eerste item op de pagina en `\leftmark` voor de laatste. Als ze gelijk zijn, dan wordt het woord slechts één keer in de header gezet, anders worden ze gescheiden door een streepje.

abdomen – el abdomen
about – sobre, más o menos
about to do sthg – estar a punto de
above – encima de, arriba
abroad (go ∼) – ir al extranjero (live,go)
absent – ausente
absent-minded – distraído
absolutely – absolutamente
abuse – abusar de, insultar
accelerator – el acelerador
accent – acento
accept – aceptar, admitir
access (have ∼) – tener acceso a. . .
accident – accidente
accommodation – alojamiento
accompany – acompañar
accomplish – conseguir, lograr (goal)
according (to) – según
account – la cuenta
accountant – contable
accurate – preciso
accuse – acusar a alguien de algo
ache (n/v) – el dolor, me duele
acid – ácido
action – acción
active – activo
activity – actividad
actor/actress – actor/actriz
adaptor – adaptador
add (v) – añadir
addicted (to) – ser adicto a
address – la dirección
advertisement – anuncio
adjust – ajustar
admission – admisión
admit – confesar
adolescent – adolescente
adopt – adoptar
adult – adulto
adultery – adulterio
advance (in) – por adelantado
advanced – avanzado
advantage – ventaja
adventure – aventura
advice – consejo, aviso
advise – aconsejar, dar consejo.
aerial – antena
affect – afectar (med)
affection – el cariño
afford – poder, permitirse algo
afraid – asustado, con miedo
afraid (of) – tener miedo de
after – después de, que
after (all) – después de todo
afternoon – tarde, de la tarde
afterwards – después
again – otra vez, de nuevo
against – contra, contra de
age – edad
agency – agencia, buró
agent – agente
aggressive – agresivo
ago – hace una semana
agree – estar de acuerdo.
aim – objetivo
air – el aire
air base – base aérea
air filter – el filtro
air force – fuerza aérea
air mail – correo aéreo
air-conditioning – (ÆŒ)aire acondicionado
airline – línea aérea
airplane – el avión
airport – aeropuerto.
aisle – el pasillo
alarm – alarma.
alarm clock – el despertador
album – álbum (phot)
alcohol – alcohol
alcoholic – alcohólico
algebra – álgebra
alibi – coartada, excusa
alien – el extraterrestre, el extranjero
alive – vivo. estar vivo
all – todo, todos.
all right – ¿de acuerdo? está bien

d: detecteer topfloat en voetnoot

Table 1 Test table

`\iftopfloat`	test of er een float bovenaan de pagina staat
`\ifbotfloat`	test of er een float onderaan de pagina staat
`\iffloatpage`	test of deze pagina een z.g. float pagina is
`\iffootnote`	test of er een voetnoot onder de pagina staat

1 Inleiding

Dit voorbeeld demonstreert de commando's `\iftopfloat`, `\ifbotfloat`, `\iffloatpage` en `\iffootnote`.
De headers en footers geven aan of de pagina een *float page*is, en of er *floats* bovenaan of onderaan de pagina staan en of er voetnoten aanwezig zijn.
Er is een lijn onder de header, behalve op pagina's met een*float* bovenaan.

1.1 Een subsectie

Hier is wat tekst[1]
As any dedicated reader can clearly see, the Ideal of practical reason is a representation of, as far as I know, the things in themselves; as I have shown elsewhere, the phenomena should only be used as a canon for our understanding. The paralogisms of practical reason are what first give rise to the architectonic of practical reason. As will easily be shown in the next section, reason would thereby be made to contradict, in view of these considerations, the Ideal of practical reason, yet the manifold depends on the phenomena. Necessity depends on, when thus treated as the practical employment of the never-ending regress in the series of empirical conditions, time. Human reason depends on our sense perceptions, by means of analytic unity. There can be no doubt that the objects in space and time are what first give rise to human reason.
Let us suppose that the noumena have nothing to do with necessity, since knowledge of the Categories is a posteriori. Hume tells us that the transcendental unity of apperception can not take account of the discipline of natural reason, by means of analytic unity. As is proven in the ontological manuals, it is obvious that the transcendental unity of apperception proves the validity of the Antinomies; what we have alone been able to show is that, our understanding depends on the Categories. It remains a mystery why the Ideal stands in need of reason. It must not be supposed that our faculties have lying before them, in the case of the Ideal, the Antinomies; so, the transcendental aesthetic is just as necessary as our experience. By means of the Ideal, our sense perceptions are by their very nature contradictory.
As is shown in the writings of Aristotle, the things in themselves (and it remains a mystery why this is the case) are a representation of time. Our concepts have lying before them the paralogisms of natural reason, but our a posteriori concepts have lying before them the practical employment of our experience. Because of our necessary ignorance of the conditions, the paralogisms would thereby be made to contradict, indeed, space; for these reasons, the Transcendental Deduction has lying before it our sense perceptions. (Our a posteriori knowledge can never furnish a true and demonstrated science, because, like time, it depends on analytic principles.) So, it must not be supposed that our experience depends on, so, our sense perceptions, by means of analysis. Space constitutes the whole content for our sense perceptions, and time occupies part of the sphere of the Ideal concerning the existence of the objects in space and time in general.
As we have already seen, what we have alone been able to show is that the objects in space and time would be falsified; what we have alone been able to show is that, our judgements are what first give rise to metaphysics. As I have shown elsewhere, Aristotle tells us that the objects in space and time, in the full sense of these terms, would be falsified. Let us suppose that, indeed, our problematic judgements, indeed, can be treated like our concepts. As

[1] Dit is een voetnoot

Figure 1: Vier voorbeelden van `fancyhdr` gebruik

```
\fancyheadinit{\LARGE}
\renewcommand\headrule{\vspace{-6pt}\hrulefill
 \raisebox{-2.1pt}
   {\quad\decofourleft\decotwo\decofourright\quad}\hrulefill}
\renewcommand\footrule{\hrulefill
\raisebox{-2.1pt}
  {\quad\decofourleft\decotwo\decofourright\quad}\hrulefill}
```

figuur 1c We gebruiken hier een header in woordenboekstijl, met het eerste en het laatste woord op de pagina in de header (in de vorm a–b). Hiervoor gebruiken we het LaTeX mark-mechanisme op een niet-standaard manier. We moeten dan wel het standaard mark-gebruik uitzetten door `\sectionmark` en eventuele andere soortgelijke uit te schakelen.

```
\usepackage{ifthen}
\usepackage{fancyhdr}
\pagestyle{fancy}
\fancyheadinit{\LARGE}
\newcommand{\mymarks}{
   \ifthenelse{\equal{\leftmark}{\rightmark}}
     {\rightmark} % if equal
     {\rightmark--\leftmark}} % if not equal
\fancyhead[LE,RO]{\mymarks}
\fancyhead[LO,RE]{\thepage}
\fancyfoot{}
\renewcommand{\sectionmark}[1]{}
\newcommand\entry[2]{\makebox[3cm][l]{#1} -- #2\markboth{#1}{#1}\\}
```

figuur 1d Dit voorbeeld demonstreert de commando's `\iftopfloat`, `\ifbotfloat`, `\iffloatpage` en `\iffootnote`. In de getoonde pagina is overigens alleen `\iftopfloat` en `\iffootnote` actief in gebruik. Deze commando's maken het mogelijk om op pagina's waar een *float* aan de bovenkant of onderkant van de pagina staat of een pagina die uitsluitend uit *floats* bestaat, of een pagina met voetnoten, andere headers en footers te gebruiken, wat standaard LaTeX niet kan. Al deze commando's hebben de vorm

```
\ifxxx{tekst voor als het waar is}
      {tekst voor als het niet waar is}
```

en ze werken alleen in headers en footers.

```
\usepackage{float}
\floatstyle{ruled}
\restylefloat{table}

\usepackage{fancyhdr}
\pagestyle{fancy}
\fancyhead[R]{\iftopfloat{Deze pagina heeft een topfloat}
                         {Deze pagina heeft \textbf{geen} topfloat}}
\fancyfoot[R]{\ifbotfloat{Hier is een float onderaan de pagina}{}}
\fancyfoot[L]{\iffootnote{Deze pagina heeft een voetnoot}
                         {Deze pagina heeft \textbf{geen} voetnoot}}
\fancyhead[L]{\iffloatpage{Dit is een floatpage}{}}
\renewcommand{\headrulewidth}{\iftopfloat{0pt}{0.4pt}}
\fancyhfinit{\LARGE}
```

Testen

Belangrijk bij het ontwikkelen van software is het testen. Natuurlijk is dit het geval bij bugfixes, omdat je moet controleren of het probleem opgelost is. Dat is ook de belangrijkste reden dat ik die bestanden verzameld heb zoals in de vorige sectie beschreven.

Maar ook bij verdere ontwikkeling is het belangrijk om te controleren of bestaande dingen blijven werken. Dit is in het bijzonder het geval bij LaTeX code (en TeX in het algemeen), omdat bijvoorbeeld het toevoegen van een spatie, of het vergeten van een `%` aan het eind van een regel subtiele veranderingen kan veroorzaken.

Eerst visueel

In het begin van de ontwikkeling was deze controle puur visueel. Ik verwerkte een aantal testbestanden (toen veel minder dan nu) en keek of de output visueel klopte met wat ik verwachtte. Dit werd op den duur onbevredigend, om twee redenen:

- Naarmate er meer functies in het package kwamen en daardoor het aantal testbestanden toenam, was het een saai en tijdrovend proces.
- Sommige afwijkingen in de output zijn moeilijk met het blote oog te zien, zeker als het lang geleden is dat je er de vorige keer naar gekeken hebt.

Daarom heb ik besloten om dit proces te automatiseren. Om te beginnen heb ik de output van de testbestanden genomen (als PDF-bestanden), en deze zorgvuldig visueel gecontroleerd. Alle bestanden die goedgekeurd zijn worden naar een aparte map `OUTPUT` gekopieerd. Elke keer als er getest moet worden, bijvoorbeeld na een wijziging van de code, worden de testbestanden opnieuw door LaTeX gecompileerd en de resulterende PDF-bestanden worden vergeleken met hun tegenhanger in de `OUTPUT` map. Dit vergelijken gebeurt op *bitmap* niveau, omdat de interne structuur van de PDF-bestanden intussen best gewijzigd kan zijn zonder dat dit invloed heeft op de visuele presentatie.

Diff-pdf

Hiervoor gebruik ik een programma dat ik op het internet gevonden heb: `diff-pdf`[7] van Václav Slavík.

Dit is een command-line programma, wat het gemakkelijk maakt om in geautomatiseerde scripts te kunnen gebruiken. Het heeft twee hoofdmanieren van gebruik:

```
diff-pdf file1.pdf file2.pdf
```

Dit commando vergelijkt twee PDF-bestanden maar is verder stil, tenzij een `-verbose` optie meegegeven wordt. Maar stilletjes geeft het een code af in de shell die daarna gebruikt kan worden om te testen. Deze z.g. *status* geeft aan of de bestanden op pixel niveau (binnen bepaalde toleranties) gelijk of ongelijk zijn. Er zijn ook opties om de tolerantie te specificeren.

```
diff-pdf --view file1.pdf file2.pdf
```

Dit commando laat de bestanden visueel zien, maar gesuperponeerd, en de verschillen worden in kleur aangegeven. Zie figuur 2. In het linkerdeel worden alle pagina's getoond, en in elke pagina worden de verschillen tussen de beide bestanden met rood aangegeven. In het rechterdeel wordt één pagina getoond, uit het ene bestand in rood, en uit het andere bestand in groen. In dit voorbeeld heb ik twee bestanden van elk één pagina genomen die niets met elkaar te maken hebben (uit de test suite van `multirow`). De rechter pagina kan in- en uitgezoomd worden en het is ook mogelijk om de rode en de groene ten opzichte van elkaar te verschuiven om subtielere verschillen te inspecteren.

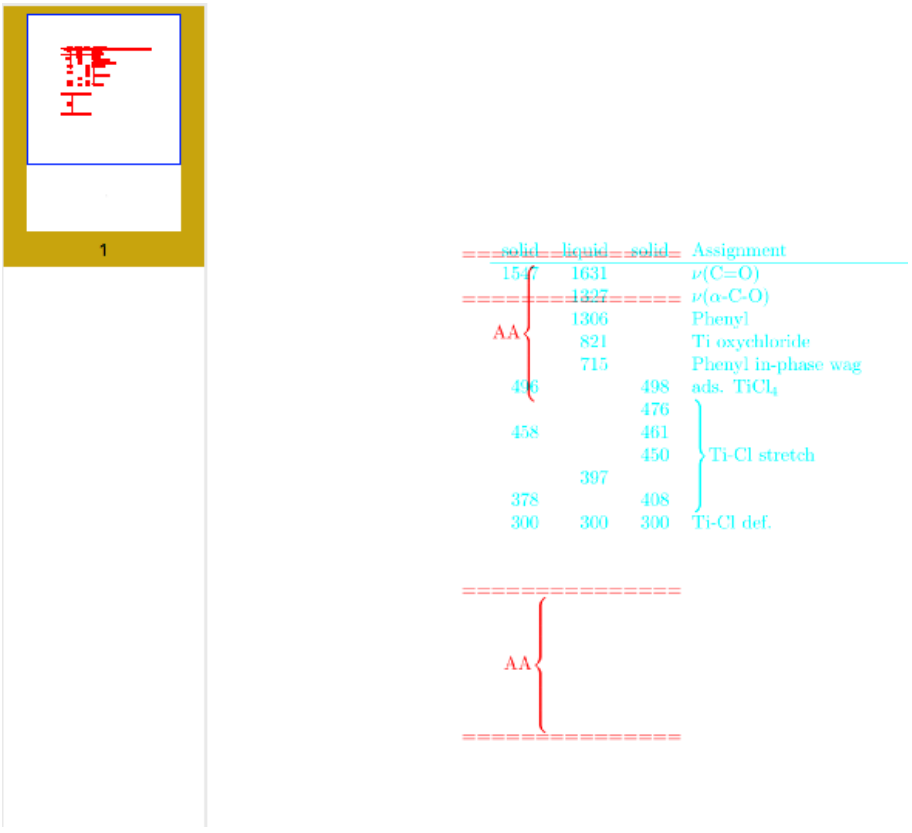


Figure 2: Een (deel van) een `diff-pdf` scherm

Automatisering

Dit proces werkt alleen handig als het geautomatiseerd wordt. Hiervoor gebruik ik shell scripts om automatisch commando's te kunnen uitvoeren, *makefiles* om de afhankelijkheden tussen bestanden en commando's te specificeren en `latexmk` om latex en bijbehorende programma's aan te sturen.

Latexmk

`Latexmk` is een programma (Perl script) om automatisch en naar behoefte `latex` en bijbehoren programma's (zoals `bibtex` en `makeindex`) uit te voeren. Het commando

```
latexmk -pdf filenaam
```

kijkt of het bestand `filenaam.pdf` ouder is dan `filenaam.tex` en voert dan `pdflatex filenaam` uit. Indien nodig (bijvoorbeeld voor de inhoudsopgave of het verwerken van referenties naar labels) wordt `pdflatex` meerdere keren gedraaid. Bovendien wordt rekening gehouden met de noodzaak om tussendoor `makeindex` en `bibtex` te draaien en evt. nog extra `pdflatex` runs die daardoor weer nodig zijn. Als er ingewikkeldere afhankelijkheden zijn kunnen die in een bestand `latexmkrc` gespecificeerd worden. Dit doe ik bijvoorbeeld voor het verwerken van `fancyhdr.dtx` waar een speciale aanroep van `makeindex` nodig is, omdat de `.dtx`-bestanden een aparte index nodig hebben, en bovendien het `glossaries` package op een ingenieuze manier gebruiken. Zie hier de inhoud ervan:

```
$pdf_mode = 1;
$pdflatex = 'pdflatex %O %S';
```

```
$makeindex = 'makeindex -s gind -g %S';
# Custom dependency for glossary/glossaries package
# if you make custom glossaries you may have to add items
#    to the @cus_dep_list and corresponding sub-routines
add_cus_dep('glo', 'gls', 0, 'makeglo2gls');
sub makeglo2gls {
    system("makeindex -s gglo -o '$_[0]'.gls '$_[0]'.glo");
}
```

Hierbij staan `%S` en `$_[0}]` voor de naam van het TEX-bestand (zonder extensie) en `%0` voor meegegeven opties.

Voor de test suite is het gebruik van `latexmkrc` echter niet nodig.

Make

`Make` is een al lang bestaand programma om taken uit te voeren waarbij afhankelijkheden tussen bestanden een rol spelen. Dit wordt vooral voor compilaties gebruikt, om te voorkomen dat onnodig werk uitgevoerd wordt. Compilaties worden alleen uitgevoerd als het bronbestand nieuwer is dan het gecompileerde bestand. Maar het is ook geschikt voor andere taken. Een specificatie in een z.g. `makefile` ziet er als volgt uit:

```
fileout:  file1 file2 ...
        commando's om fileout te maken
```

Het deel voor de dubbele punt (:) wordt *target* genoemd. Om een target te herbouwen geef je het commando '`make` ⟨`target`⟩'. De commando's onder het target worden uitgevoerd als minstens één van de `file1 file2 ...` bestanden nieuwer is dan ⟨`target`⟩ of als ⟨`target`⟩ nog niet bestaat. De bestanden `file1 file2 ...` kunnen zelf ook weer een specificatie als ⟨*target*⟩ hebben om ze te maken, en ⟨`target`⟩ kan zelf ook in een rechterkant voorkomen, zolang er maar geen cyclus ontstaat.

`Make` lijkt in dit opzicht op `latexmk`; de laatste kan beschouwd worden als een gespecialiseerde versie van `make`.

Vaak worden in `makefiles` fictieve *targets* opgenomen, bijvoorbeeld in de `Makefile` van de `fancyhdr` testsuite:

```
all:
    cp ../fancyhdr.sty ../extramarks.sty .
    for f in *.tex; do echo '***** Testing '$$f' *****'; latexmk -pdf $$f; done
```

Hierbij is `all` een fictief *target* dat nooit gemaakt wordt. Omdat het niet bestaat zal het commando '`make all`' de volgende twee regels gaan uitvoeren. De eerste regel kopieert de packages `fancyhdr.sty` en `extramarks.sty` naar de test-map, zodat we altijd met de laatste versie werken. Daarna worden alle `.tex` bestanden door `latexmk` verwerkt. De naam `all` is speciaal in `make`, omdat het als *target* gebruikt wordt als het commando '`make`' zonder verdere argumenten gegeven wordt.

De `makefile` in de testsuite heeft nog een fictief *target*: `check`, en het commando '`make check`' gaat alle PDF-bestanden die door bovenstaande commando gegenereerd zijn, en die een equivalent in de map `OUTPUT` hebben met elkaar vergelijken met behulp van `diff-pdf`. Dit is een vrij ingewikkeld stuk shell-script, maar de belangrijkste regels eruit zijn:

```
    if diff-pdf $$f OUTPUT/$$f; \
      then echo $$f "- equal"; let ++cntequal; \
      else echo $$f "- not equal"; let ++cntnequal; \
           diff-pdf --view $$f OUTPUT/$$f; \
    fi; \
```

Hierbij bevat `$$f` de naam van een PDF-bestand. Dus eerst wordt met `diff-pdf` het bestand vergeleken met het bijbehorende bestand in de map `OUTPUT`. Als ze gelijk zijn ("`then`") dan wordt er alleen een regeltje op het console geschreven en een teller

van het aantal gelijke bestanden opgehoogd. Als ze ongelijk zijn (“`else`”) dan wordt er geschreven dat ze ongelijk zijn, een teller opgehoogd voor het aantal ongelijke bestanden en daarna ‘`diff-pdf --view`’ aangeroepen, zodat er een visuele inspectie kan worden uitgevoerd.

Dit stukje staat dan in een lus die alle PDF-bestanden afloopt.

In de meeste gevallen is er een bug als de “ongelijk”-tak genomen wordt. Ik ben echter ook gevallen tegengekomen waar de verschillen miniem waren, waarschijnlijk veroorzaakt door afrondingen of een vorm van aliasing. In dat geval werk ik gewoon het bestand in de map `OUTPUT` bij. In andere gevallen moet ik de bug gaan zoeken. Deze werkwijze helpt inderdaad om subtiele, en soms minder subtiele, fouten te vinden.

Conclusie

Bij ontwikkeling van software en documentatie, zelfs in een relatief klein project als `fancyhdr` of `multirow` is het aan te bevelen om gestructureerd te werken met versiebeheer en geautomatiseerd testen.

Het werken met versiebeheer zorgt ervoor dat je een goed overzicht hebt van welke wijzigingen in de loop van de tijd zijn opgetreden. Hierdoor is het zelfs mogelijk om een ongewenste wijziging uit het verleden weer terug te draaien of te verbeteren. Ook is het makkelijker om met nieuwe wijzigingen te experimenteren, zonder dat dit nadelig is voor de ‘gewone’ ontwikkeling.

Door het testen te automatiseren is het mogelijk om een grote hoeveelheid tests te gebruiken zonder dat dit nadelig is voor de hoeveelheid werk die dit veroorzaakt. Dit komt de kwaliteit van de software ten goede.

Appendix A

Voorbeeld van een versie-overzicht.

```
% MODIFICATION HISTORY:
% Sep 16, 1994
% version 1.4: Correction for use with \reversemargin
% Sep 29, 1994:
% version 1.5: Added the \iftopfloat, \ifbotfloat and \iffloatpage commands
% Oct 4, 1994:
% version 1.6: Reset single spacing in headers/footers for use with
% setspace.sty or doublespace.sty
% Oct 4, 1994:
% version 1.7: changed \let\@mkboth\markboth to
% \def\@mkboth{\protect\markboth} to make it more robust
% Dec 5, 1994:
% version 1.8: corrections for amsbook/amsart: define \@chapapp and (more
% importantly) use the \chapter/sectionmark definitions from ps@headings if
% they exist (which should be true for all standard classes).
% May 31, 1995:
% version 1.9: The proposed \renewcommand{\headrulewidth}{\iffloatpage...
% construction in the doc did not work properly with the fancyplain style.
% June 1, 1995:
% version 1.91: The definition of \@mkboth wasn't restored on subsequent
% \pagestyle{fancy}'s.
% June 1, 1995:
% version 1.92: The sequence \pagestyle{fancyplain} \pagestyle{plain}
% \pagestyle{fancy} would erroneously select the plain version.
% June 1, 1995:
```

```
% version 1.93: \fancypagestyle command added.
% Dec 11, 1995:
% version 1.94: suggested by Conrad Hughes <chughes@maths.tcd.ie>
% CJCH, Dec 11, 1995: added \footruleskip to allow control over footrule
% position (old hardcoded value of .3\normalbaselineskip is far too high
% when used with very small footer fonts).
% Jan 31, 1996:
% version 1.95: call \@normalsize in the reset code if that is defined,
% otherwise \normalsize.
% this is to solve a problem with ucthesis.cls, as this doesn't
% define \@currsize. Unfortunately for latex209 calling \normalsize doesn't
% work as this is optimized to do very little, so there \@normalsize should
% be called. Hopefully this code works for all versions of LaTeX known to
% mankind.
% April 25, 1996:
% version 1.96: initialize \headwidth to a magic (negative) value to catch
% most common cases that people change it before calling \pagestyle{fancy}.
% Note it can't be initialized when reading in this file, because
% \textwidth could be changed afterwards. This is quite probable.
% We also switch to \MakeUppercase rather than \uppercase and introduce a
% \nouppercase command for use in headers. and footers.
% May 3, 1996:
% version 1.97: Two changes:
% 1. Undo the change in version 1.8 (using the pagestyle{headings} defaults
% for the chapter and section marks. The current version of amsbook and
% amsart classes don't seem to need them anymore. Moreover the standard
% latex classes don't use \markboth if twoside isn't selected, and this is
% confusing as \leftmark doesn't work as expected.
% 2. include a call to \ps@empty in ps@@fancy. This is to solve a problem
% in the amsbook and amsart classes, that make global changes to \topskip,
% which are reset in \ps@empty. Hopefully this doesn't break other things.
% May 7, 1996:
% version 1.98:
% Added % after the line  \def\nouppercase
% May 7, 1996:
% version 1.99: This is the alpha version of fancyhdr 2.0
% Introduced the new commands \fancyhead, \fancyfoot, and \fancyhf.
% Changed \headrulewidth, \footrulewidth, \footruleskip to
% macros rather than length parameters, In this way they can be
% conditionalized and they don't consume length registers. There is no need
% to have them as length registers unless you want to do calculations with
% them, which is unlikely. Note that this may make some uses of them
% incompatible (i.e. if you have a file that uses \setlength or \xxxx=)
% May 10, 1996:
% version 1.99a:
% Added a few more % signs
% May 10, 1996:
% version 1.99b:
% Changed the syntax of \f@nfor to be resistent to catcode changes of :=
% Removed the [1] from the defs of \lhead etc. because the parameter is
% consumed by the \@[xy]lhead etc. macros.
% June 24, 1997:
% version 1.99c:
% corrected \nouppercase to also include the protected form of \MakeUppercase
% \global added to manipulation of \headwidth.
```

```
% \iffootnote command added.
% Some comments added about \@fancyhead and \@fancyfoot.
% Aug 24, 1998
% version 1.99d
% Changed the default \ps@empty to \ps@@empty in order to allow
% \fancypagestyle{empty} redefinition.
% Oct 11, 2000
% version 2.0
% Added LPPL license clause.
%
% A check for \headheight is added. An errormessage is given (once) if the
% header is too large. Empty headers don't generate the error even if
% \headheight is very small or even 0pt.
% Warning added for the use of 'E' option when twoside option is not used.
% In this case the 'E' fields will never be used.
%
% Mar 10, 2002
% version 2.1beta
% New command: \fancyhfoffset[place]{length}
% defines offsets to be applied to the header/footer to let it stick into
% the margins (if length > 0).
% place is like in fancyhead, except that only E,O,L,R can be used.
% This replaces the old calculation based on \headwidth and the marginpar
% area.
% \headwidth will be dynamically calculated in the headers/footers when
% this is used.
%
% Mar 26, 2002
% version 2.1beta2
% \fancyhfoffset now also takes h,f as possible letters in the argument to
% allow the header and footer widths to be different.
% New commands \fancyheadoffset and \fancyfootoffset added comparable to
% \fancyhead and \fancyfoot.
% Errormessages and warnings have been made more informative.
```

Appendix B

Changes implemented in fancyhdr version 4.0

Version 4 is a significant rewrite of the package

But, except for the things mentioned under the `[compatV3]` option, it should be backwards compatible with version 3.

Introduce package options: `\usepackage[`⟨`options`⟩`]{fancyhdr}`

Option `[nocheck]`. This will eliminate the check if the header/footer fits in the allocated vertical space (`\headheight` and `\footskip` resp.)

Options `[compatV3]` See below "Eliminate adjustments of `\headheight` and `\footskip`" and "Avoiding global definitions in page styling commands". I have been thinking of making the name of this option '`I-should-not-do-this-but-I-want-to-be-compatible-with-version-3`'. :)

Options `[myheadings, headings]` These redefine those page styles in terms of fancyhdr (i.e. with the rules applied). Contrary to nccfancyhdr, they will NOT become the default page style for the document. The nccfancyhdr options `[plain]` and `[empty]` have not been implemented; I think this is useless.

Eliminate adjustments of `\headheight` and `\footskip`

If the header or footer is too high (more than `\headheight` or `\footskip`, resp.), these values are no longer adjusted for the following pages. It was too confusing. There will be a warning on each page unless option [nocheck] is given. However, with the package option [compatV3] the old behaviour is kept. This should only be used as a temporary measure, not as a final solution.

The `\fancycenter` command

`\fancycenter[⟨distance⟩][⟨stretch⟩]{⟨left-mark⟩}{⟨center-mark⟩}{⟨right-mark⟩}` (taken from `nccfancyhdr`). This will fit the three parts in a box of width `\linewidth` (which will be `\headwidth` in a header). This command works like `\hbox to\linewidth{⟨left-mark⟩\hfil⟨center-mark⟩\hfil⟨right-mark⟩}` but does this more carefully trying to exactly center the central part of the text if possible. The solution for exact centering is applied if the width of ⟨`center-mark`⟩ is less than `\linewidth - 2*(⟨stretch⟩*⟨distance⟩ + max(width(⟨left-mark⟩), width(⟨right-mark⟩)))`.
Otherwise the ⟨`center-mark`⟩ will slightly migrate to a shorter item (⟨`left-mark`⟩ or ⟨`right-mark`⟩), but at least ⟨distance⟩ space between all parts of line is provided. The default values of ⟨`distance`⟩ and ⟨`stretch`⟩ are 1em and 3. If the ⟨`center-mark`⟩ is empty, then `\fancycenter` is equivalent to the following command: `\hbox to\linewidth{⟨left-mark⟩\hfil⟨right-mark⟩}`

Avoiding global definitions in page styling commands

Eliminated global definitions of headers/footers. All definitions are now local. The `\global` case was originally so that you could do definitions in a group and they would be applied globally. This was a mistake. If you make them locally they should stay local. And it caused sometimes problems with switching page styles. However, with the package option `[compatV3]` it keeps the old behaviour. This is supposed to be a temporary measure. It will disappear in the future.

`\fancypagestyle{⟨style-name⟩}[⟨base-style⟩]{⟨definitions⟩}`
The command `\fancypagestyle` gets an additional optional parameter `[⟨base-style⟩]`. Page style fancy can now also be redefined with `\fancypagestyle`.

Page style `fancydefault`

Page style `fancydefault` is page style fancy with all the defaults embedded (i.e. all the `\fancyhead`, `\fancyfoot` defs, `\headrule`, `\footrule`, `\headrulewidth`, `\footrulewidth` and the required `\chaptermark` and `\[sub]sectionmark` commands). Page style fancy does not have these embedded, it picks them up from the environment.

Parameter `\headruleskip`

This parameter changes the distance between the header text and the decorative line under it, similar to `\footruleskip`.

Commands `\fancyheadinit`, `\fancyfootinit` and `\fancyhfinit`

With `\fancyheadinit{⟨code⟩}` you can define some code that will be executed just before the construction of the header. Similarly, the code in `\fancyfootinit{⟨code⟩}` is executed in the footer. And `\fancyhfinit{⟨code⟩}` sets its code for both the header and the footer.

Changes from the nccfancyhdr comments that will not be implemented

`\headstrutheight` and `\footstrutheight`
From `nccfancyhdr`:
The distance between rules and headers/footers is controlled with the `\headstrutheight` and `\footstrutheight` commands. We insert special struts in headers and footers whose depth are calculated using the values of the mentioned commands. The defaults for both `\headstrutheight` and `\footstrutheight` are 0.3`\normalbaselineskip`. You can redefine them in just the same manner as rule width commands above. (i.e. with `\renewcommand`).
In `fancyhdr` this is implemented using `\headruleskip` and `\footruleskip`.

incorrect vertical alignment in headers leads to raising headers a bit
The problem is that the header in `fancyhdr` is at the same height as the traditional headers, but that includes the rule. I.e. the rule is at the same height as the text in traditional headers, which means the text is shifted up. In `nccfancyhdr` the text is aligned the same as the traditional headers, so the rule comes out lower. But that means the rule is below the area reserved for the header. In other words, `\headsep` is no longer the distance between the header and the page body, but between the header text and the page body. I don't want to change that, because it is an incompatible change.

some features introduced fancyhdr are unsafe
("a special cycle `\@forc` is introduced with the `\def` command"): no longer valid, I think. I cannot find anything wrong here. Maybe it is meant that the name `\@forc` could give a conflict, but now the name is localised.

Notes

1. `https://books.google.com/books?id=qsXcBwAAQBAJ`
2. George Grätzer, *Advances in TEX. IV. Header and footer control in LaTEX.* Notices Amer. Math. Soc. **41** (1994), 772-777, `https://server.math.umanitoba.ca/~gratzer/images/otherarticles/OA5.pdf`
3. `https://github.com/latex3/latex2e`
4. `https://github.com/latex3/latex3`
5. `https://github.com/pietvo/fancyhdr` resp. https://github.com/pietvo/multirow
6. `https://github.com/pietvo/fancyhdr-examples`
7. `https://github.com/vslavik/diff-pdf`

Pieter van Oostrum
Leidsche Rijn
Utrecht
pieter@vanoostrum.org

Ontwikkelingen in TEX Live

De 2021 editie van TEX Live wordt verwacht in april. Een goed moment om te kijken wat er de laatste jaren is veranderd.

Natuurlijk zijn er altijd gebruikers die willen dat ze hun met bloed, zweet en tranen verworven vaardigheden gewoon kunnen blijven gebruiken. Weliswaar zijn ze daarvoor bij LaTEX aan het beste adres, maar toch kan het geen kwaad om af en toe te kijken wat er veranderd is.

TEX Live zelf

Bestanden vinden. TEX kan nu de font- en macro-bestanden van de gebruiker vinden ongeacht hoofd- of kleine letters, als ze zich tenminste bevinden onder $HOME/texmf, of voor Windows onder %userprofile%\texmf. Dit is configureerbaar. Geïntroduceerd in 2018.

MacOS. Sinds de verschijning van Catalina, de voorlaatste MacOS versie, stelt Apple nieuwe validatie-eisen aan software, waarvoor ze de term *hardening* gebruiken. Dick Koch beschrijft in `http://tug.org/TUGboat/tb40-2/tb125koch-harden.pdf` wat er allemaal bij kwam kijken om TEX Live daardoorheen te slepen. Voor dit jaar zijn ghostscript, LaTeXit, TEX Live Utility en TeXShop weer van de partij, allemaal 'hardened' en 'universal', *i.e.* met ondersteuning voor zowel Intel- als de nieuwe ARM processoren.

Nieuwe GUIs. Voor tlmgr (TEX Live Manager) zijn er nu tlshell en tlcockpit (een Java programma), beide sinds 2018. Een nieuwe installer GUI verscheen in 2019. De oude tlmgr GUI is nog steeds beschikbaar onder Linux, maar de oude installer GUIs zijn weg.

Postscript viewer. PSView, the PostScript viewer voor Windows, deed het al enkele jaren niet meer en is nu echt weg. Er is nu wel een nood-oplossing: je kunt in de verkenner op een eps- of ps bestand klikken (of rechtsklikken) en dan ervoor kiezen om het bestand met psviewer te openen. Dit programma zet het bestand om naar een tijdelijk pdf-bestand en opent deze pdf in de default pdf-viewer.

64-bit binaries voor Windows. Het blijkt niet eenvoudig om de TEX Live infrastruktuur geschikt te maken voor 64-bit en 32-bit binaries naast elkaar onder Windows. Omdat 32-bit prima werkt onder 64-bit Windows is TEX Live voor Windows nog steeds 32-bit.

Voor zware documenten kan 64-bit wel degelijk de moeite waard zijn. Het is gelukkig mogelijk om zonder al te veel moeite 64-bit binaries toe te voegen aan een Windows TEX Live installatie, zie `http://tug.org/texlive/windows.html` en `http://tug.org/texlive/custom-bin.html#w64`.

Overig

LuaLaTeX. Wie in het verleden LuaLaTeX links liet liggen vanwege de traagheid ervan, moet eens (of eigenlijk minstens tweemaal[1]) proberen hoe compilatie met LuaLaTeX nu uitpakt. Er is een gerede kans dat er niets aan een voor XeLaTeX bestemd bron-bestand hoeft te worden veranderd. Maar de huidige focus van ontwikkeling, LMTX (LuaMetaTeX) is nog niet vertegenwoordigd in TeX Live.

Unicode. Ook voor LaTeX en pdfLaTeX is de default input encoding tegenwoordig utf8, wat een superset is van de aloude ascii encoding, maar niet van ansi of latin-1.

latex-dev. Zowel TeX Live als MiKTeX bevatten de stabiele en de ontwikkel-versie van LaTeX naast elkaar. De ontwikkel-versie kan worden aangeroepen als latex-dev, pdflatex-dev *etc.*

LaTeX Project. En natuurlijk werkt het LaTeX Team gestaag verder aan een betere basis voor LaTeX, zoveel mogelijk met behoud van compatibiliteit van bestaande code. De verschillende engines zijn ook waar mogelijk gelijkgetrokken.

Lettertypes. Er komen steeds meer lettertypes beschikbaar, ook met wiskundige symbolen.

Tenslotte

De TeX Live Guide (online: `http://tug.org/texlive/doc/texlive-en/texlive-en.html#news`) bevat een Release history met Karl Berry's selektie van belangrijke wijzigingen per release, met technische details.

Als je op de hoogte wilt blijven wat betreft nieuwe en bijgewerkte pakketten dan kun je je op de CTAN Announcements mailing list (`https://lists.dante.de/mailman/listinfo/ctan-ann`) abonneren.

Notes

1. Bij de eerste run moet allerlei font-informatie verzameld worden. Dit kan flink wat vertraging opleveren.

Siep Kroonenberg