TUGboat

Volume 16, Number 1 / March 1995

	3	Addresses
	4	EuroTEX94 Contest Answers / *Barbara Beeton*
General Delivery	5	Opening words / *Christina Thiele* and *Michel Goossens*
	8	Editorial comments / *Barbara Beeton*
Software & Tools	9	Making `MakeTeXPK` safer for Unix installations / *Michael Jaegermann*
	12	Hyphenation exception log / *Barbara Beeton*
Philology	18	Configuring TEX or LATEX for typesetting in several languages / *Claudio Beccari*
	30	How to make a foreign language pattern file: Romanian / *Claudio Beccari, Radu Oprea* and *Elena Tulei*
	42	TEX and Linguistics / *Christina Thiele*
Font Forum	45	Introducing METAPOST / *Alan Hoenig*
	46	Some METAFONT techniques / *Yannis Haralambous*
	54	The program `a2ac` — Font handling on the PostScript level / *Petr Olsak*
	60	Problems of the conversion of METAFONT fonts to PostScript Type 1 / *Basil Malyshev*
	69	Partial font embedding utilities for PostScript Type-1 fonts / *Basil Malyshev* and *Michel Goossens*
	78	Tight setting with TEX / *Alan Jeffrey*
LATEX	80	XƎMTEX for drawing chemical structural formulas / *Shinsaku Fujita*
News & Announcements	89	Calendar
	90	Production notes / *Mimi Burbank*
	91	Coming next issue
TUG Business	92	Institutional members
Forms	93	TUG membership application
Advertisements	94	TEX consulting and production services

TEX Users Group

Memberships and Subscriptions

TUGboat (ISSN 0896-3207) is published quarterly by the TEX Users Group, Flood Building, 870 Market Street, #801; San Francisco, CA 94102, U.S.A.

1995 dues for individual members are as follows:

- Ordinary members: $55 with *TUGboat*; $40 without;
- Students: $35 with *TUGboat*; $20 without.

Membership in the TEX Users Group is for the calendar year, and includes all issues of *TEX and TUG NEWS* for the year in which membership begins or is renewed. *TUGboat* may be included for a supplementary fee. Individual membership is open only to named individuals, and carries with it such rights and responsibilities as voting in the annual election. A membership form is provided on page 93.

TUGboat subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. Subscription rates: North America $60 a year; all other countries, ordinary delivery $60, air mail delivery $80.

Second-class postage paid at San Francisco, CA, and additional mailing offices. Postmaster: Send address changes to *TUGboat*, TEX Users Group, 1850 Union Street, #1637, San Francisco, CA 94123, U.S.A.

Institutional Membership

Institutional Membership is a means of showing continuing interest in and support for both TEX and the TEX Users Group. For further information, contact the TUG office.

Printed in U.S.A.

Addresses

All correspondence, payments, etc.
TEX Users Group
1850 Union Street, #1637
San Francisco,
CA 94123 USA

Parcel post, delivery services:
TEX Users Group
Flood Building
870 Market Street, #801
San Francisco,
CA 94102, USA

Telephone

+1 415 982-8849

Fax

+1 415 982-8559

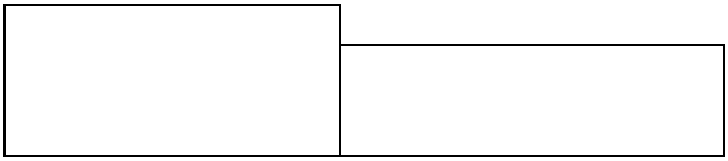
Electronic Mail (Internet)

General correspondence:
`TUG@tug.org`

Submissions to *TUGboat*:
`TUGboat@Math.AMS.org`

Who were the first compositors? Unfortunately we are not quite sure. Many of them were probably scribes faced with unemployment as a result of the new technology.

Alexander Lawson
The Compositor as Artist, Craftsman, and Tradesman (1990)



COMMUNICATIONS OF THE TEX USERS GROUP

EDITOR BARBARA BEETON

VOLUME 16, NUMBER 1 • MARCH 1995

SAN FRANCISCO • CALIFORNIA U.S.A.

TUGboat

During 1995, the communications of the TEX Users Group will be published in four issues. One issue (Vol. 16, No. 3) will contain the Proceedings of the 1995 TUG Annual Meeting. One issue (Vol. 16, No. 2) will be a theme issue, edited by a guest editor; participation will be by invitation.

TUGboat is available at a special subscription rate to all members.

Submissions to *TUGboat* are reviewed by volunteers and checked by the Editor before publication. However, the authors are still assumed to be the experts. Questions regarding content or accuracy should therefore be directed to the authors, with an information copy to the Editor.

Submitting Items for Publication

Two regular issues will be prepared in 1995. Deadlines for these and other future issues are listed in the Calendar, page 89.

Manuscripts should be submitted to a member of the *TUGboat* Editorial Board. Articles of general interest, those not covered by any of the editorial departments listed, and all items submitted on magnetic media or as camera-ready copy should be addressed to the Editor, Barbara Beeton (see address on p. 3).

Contributions in electronic form are encouraged, via electronic mail, on magnetic tape or diskette, or made available for the Editor to retrieve by anonymous FTP; contributions in the form of camera copy are also accepted. The *TUGboat* "style files", for use with either `plain` TEX or LATEX, are available "on all good archives". For authors who have no access to a network, they will be sent on request; please specify which is preferred. For instructions, write or call the TUG office.

An address has been set up on the AMS computer for receipt of contributions sent via electronic mail: `TUGboat@math.ams.org` on the Internet.

Reviewers

Additional reviewers are needed, to assist in checking new articles for completeness, accuracy, and presentation. Volunteers are invited to submit their names and interests for consideration; write to `TUGboat@math.ams.org` or to the Editor, Barbara Beeton (see address on p. 3).

TUGboat Editorial Board

See page 3 for addresses.

Other TUG Publications

TUG publishes the series *TEXniques*, in which have appeared reference materials and user manuals for macro packages and TEX-related software, as well as the Proceedings of the 1987 and 1988 Annual Meetings. Other publications on TEXnical subjects also appear from time to time.

TUG is interested in considering additional manuscripts for publication. These might include manuals, instructional materials, documentation, or works on any other topic that might be useful to the TEX community in general. Provision can be made for including macro packages or software in computer-readable form. If you have any such items or know of any that you would like considered for publication, send the information to the attention of the Publications Committee in care of the TUG office.

TUGboat Advertising and Mailing Lists

For information about advertising rates, publication schedules or the purchase of TUG mailing lists, write or call the TUG office.

Trademarks

Many trademarked names appear in the pages of *TUGboat*. If there is any question about whether a name is or is not a trademark, prudence dictates that it should be treated as if it is. The following list of trademarks which appear in this issue may not be complete.

MS/DOS is a trademark of MicroSoft Corporation
METAFONT is a trademark of Addison-Wesley Inc.
PC TEX is a registered trademark of Personal TEX, Inc.
PostScript is a trademark of Adobe Systems, Inc.
TEX and $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX are trademarks of the American Mathematical Society.
Textures is a trademark of Blue Sky Research.
UNIX is a registered trademark of UNIX Systems Laboratories, Inc.

Sebastian Rahtz
Production Methods Group
Elsevier Science Ltd
The Boulevard
Langford Lane, Kidlington
Oxford OX5 1GB, U.K.
Sebastian.Rahtz@cl.cam.ac.uk

Tomas Rokicki
Box 2081
Stanford, CA 94309 U.S.A.
415-855-9989
rokicki@cs.stanford.edu

Chris Rowley
Open University
Walton Hall
Milton Keynes MK7 6AA
United Kingdom
c.a.rowley@open.ac.uk

Janet Sullivan
TEX Users Group
1850 Union Street, #1637
San Francisco, CA 94123 U.S.A.
+1 415 982-8449
janet@tug.org

Christina Thiele
15 Wiltshire Circle
Nepean K2J 4K9, Ontario Canada
cthiele@ccs.carleton.ca

Hermann Zapf
Seitersweg 35
D-64287 Darmstadt, Germany

Answers to the EuroTEX'94 portraits.

Jiří Zlatuška Wietse Dol Vladimir Batagelj

Klaus Lagally Éric Picheral

Michel Goossens BARBARA

Phil Taylor BEETON Bernd Raichle

Johannes Braams Friedhelm Sowa

Lutz Birkhahn Jörg Knappen

General Delivery

Opening words

Christina Thiele (Outgoing President) and
Michel Goossens (Incoming President)

This first column for 1995 marks the transition between myself and Michel Goossens, TUG's new president. I am delighted to share this column with him — and even more so to pass it on to him!

Passing the baton

While this first issue of 1995 should normally have been in your mailbox last March, thus before the spring election, the delay in getting it out to you has been such that it is more reasonable to acknowledge "real-time" events.

Having a real election for the position of TUG president was very important to me, since my own term was not the result of an election by TUG members but a make-do solution to an interim situation (although I had been elected as a board member in the first elections in 1991).

In 1991, the board decided that all board members should be elected by the membership, including the president. Malcolm Clark served as interim president for one-and-a-half years, to allow the election cycle to begin properly. However, no candidates stepped forward in the fall of 1992, and so the board was forced to find a president from amongst its own members; the result was that I took on the job of president. I have therefore viewed my role these past two-and-a-half years as being more of an administrator ("paper-generating bureaucrat" might be another term for it!), focusing on internal infrastructure more than on external leadership issues.

This latter role is what TUG now needs to focus on — indeed, one could argue that this role was needed already a year ago. Perhaps. But the climate a year ago was less calm, more agitated. Now, however, I believe Michel, as new president, has a much better context in which to be a strong leader for TUG, and by extension, to better represent TUG in the general TEX community.

Looking back

Since this is my last column, I would like to take the opportunity to look back over my past seven years on TUG's board, beginning in 1988 at the Montreal annual meeting. I've seen the board move from being an appointed group to an elected body, from a group of keen implementors and developers with little administrative involvement to one where most development work is now done outside the board, even outside TUG, and administrative issues have seemed to consume the board's time and energy. In fact, most of the faces from when I first joined the board are no longer there. But this should not surprise anyone unduly.

The TEX community of today is very different from that of 7 years ago, much less 16 years ago when TUG began. The program is mature, the users are world-wide and experienced, network access and network-based resources have increased to the point where almost everything is available electronically: help, information, sources, documentation — and fellow-users. Finding a useful role in the current electronic and economic climate is very difficult for all organizations in computer-related activities. To my mind, TUG is now swinging away from the administrative focus of recent years towards more of a collaborative and coordinating role in the community at large. This is not to say that the efforts made to provide a solid administrative infrastructure have been for naught — but they only need to be done once, and after that it's more a matter of introducing refinements and improvements. The main focus can thus move elsewhere.

I'd like to think that this is where I've most usefully expended my energies for TUG — in the occasionally unimaginative and plodding business of documentation, guidelines and general information. Drafting guidelines for the proceedings, which TUG first began publishing in 1987, was one of my first ventures — mainly to provide some sort of reference point for myself as editor of the 1988 and 1989 proceedings, and of course, to also give authors some help in preparing their submissions. The guidelines have since become a regular component in the proceedings editor's arsenal of files; and each editor has been steadily improving and revising the document over the years. Which is what good guidelines should have to undergo — growth and change — to address new situations. Similarly, the four years I spent working on TUG meetings led to drafting conference guidelines along with Peter Flynn, who had experience with the Cork 1990 meeting. Other guideline writing duties I've shared include those for presenters, vendors, joint memberships, and elections.

Last year saw the introduction of "info-sheets", short one- or two-page documents which can be useful sources of information: `tug-info.tex` (general info about TUG) and `usergrps.tex` (list of all user

groups) are two of the first ones.[1] They are presented later in this issue, for everyone's information. The source files can be found on CTAN in `tex-archive/usergrps/tug` and are updated as necessary. In the works are info-sheets on CTAN and on TEX implementations for various platforms.

TTN was the biggest project, though, that I undertook: and even then, only with the substantial and informed contributions from regular columnists such as Peter Flynn, Peter Schmitt, Jeremy Gibbons and Robert Becker, was it possible to keep up a rhythm of four issues per year for three years. That publication is now also evolving, under the new editorship of Peter Flynn.

Indeed, most all of what I've worked on in TUG has been something that others have had just as big a role in, or have taken further along the road. With the right combination of people, who can make any job seem do-able, any problem solvable, and any pleasure share-able, there's really nothing like collaborative work to make you feel useful, particularly as volunteer work doesn't bring much else!

Most of what I've learned about what they call 'people skills' and all that — it's been learned by working within TUG and the TEX community. Perhaps not always well learned, but certainly it's been the best exposure to all kinds of issues, situations, and people one could wish for. Not at all what I expected to learn when I sent in my first membership form in 1986, that's for sure, or when I attended my first meeting in 1987!

TUGboat is back!

One major infrastructure concern is of course *TUGboat* production. This issue you are reading is therefore also significant in that it marks the second of what will be a "five-step program"[2] to getting TUG back into normal contact with its own membership, and by extension, serving notice to the entire TEX community that we are alive and well and working like mad to regain our members' respect and renewed membership. It has come as no surprise that our 1995 membership figures are down from last year; a big factor has been the non-appearance of our flagship publication. A major concern has therefore necessitated a major change in production.

With this in mind, as well as the growing difficulties being experienced by the TUGboat editor to devote as much time and energy towards *TUGboat* as in the past, a change in the production environment had to be made. As president, I was able to initiate discussions on the feasibility of a team approach, and in conjunction with Barbara (*TUGboat* editor), Mimi Burbank (chair of the Publications Committee), and Michel Goossens (incoming president), we were able to quickly find a new production route to follow.

The utter dependence upon one person, Barbara Beeton, to not only edit but also deal with TEXnical production had to change.[3]. *TUGboat* 15,4 was the first result of the new production approach: a team of people working under Barbara's sharp eye, each one bringing a great deal of experience in different aspects of TEX as it now is used and understood by the community. This issue (16,1) is the first one which joins the team approach with a new production site (SCRI), and will, we believe, allow for much greater scope and flexibility in the future.

SCRI support critical

The team approach has been greatly aided by the generosity of the people at Florida State University's Supercomputer Computations Research Institute (SCRI), who have allowed us to share half of a new 4GB disc in order to undertake *TUGboat* production. Mimi Burbank deserves the credit for having made this possible; and we are deeply grateful to SCRI for the additional technical and logistical support which they have provided. The disc now makes it possible for team members to access all *TUGboat* production files, lend speedy assistance and advice on problems which inevitably arise, and generally provide a solid support group for *TUGboat*'s long-standing editor. For more details, see Barbara's column elsewhere in this issue.

TUG's new president, Michel Goossens, vowed at the recent annual meeting to see that, between now and the end of the year, members will see *TUGboat* issues appearing in very short order, to get us back to the normal schedule. You have received 15,4 (the last 1994 issue). This is 16,1. You will receive 16,2 (guest-edited by Malcolm Clark) and 16,3 (the TUG'95 Proceedings, edited by Robin Fairbairns) before the end of the calendar year; the December

[1] The idea came from a one-page overview that I picked up at the 1994 annual meeting of the LSA (Linguistic Society of America) in Boston.

[2] Five issues — five steps: 15,4; 16,1 to 16,4.

[3] And that TEXnical aspect has done nothing but become more complex with each new article — proving that *TUGboat* authors are amongst the most devilishly creative group anyone should ever have to deal with! I would hasten to reiterate a point that Barbara has repeatedly made in the past: that this is *not* the aim of *TUGboat* — to be for the TEXnically devilish! There is a desperate need for solid entry-level articles that will help all users better understand and apply TEX for all purposes — not just the generation of fantastic new code and fonts!

issue, 16,4, will be out in early 1996. The work is already well in hand; elsewhere in this issue you will find information on what's coming in 16,2 and 16,3. We are convinced this is the best solution for the current and long-term survival of both *TUGboat* and TUG: without the one, the health and viability of the other is also drawn into question — not only by current and former TUG members, but by the TEX community at large.

And with this longish message, I now pass the President's Column over to Michel Goossens, TUG's incoming president.

– – ∗ – – ∗ – – ∗ – –

Moving forward

Christina Thiele, TUG's outgoing president, has explained clearly what has happened in the last year or two, stating the facts and putting them in an objective perspective. As noted previously, the problems with *TUGboat* have, amongst other things, contributed to a drop of about 20% in the membership of TUG with respect to the 1994 figures. As I announced in my inaugural speech, I am making it my first task as new president to get *TUGboat* back on schedule. In the production team that we have set up, a real spirit of cooperation has developed, with each of the team members contributing in an area where she or he feels most comfortable or has particular expertise. I am confident that this approach can be made to last, and that *TUGboat* will now arrive on time and at regular intervals on our readers' desks.

New horizons

But is it not enough to just "carry on", saying that it'll soon be "business as usual". The world of electronic publishing does not stand still; on the contrary, it is caught in a whirlwind. If TEX wants to survive, it will have to adapt to this new and changing environment. Hypertext, HTML and SGML, PDF (Portable Document Format), publishing on the Net, document re-use, CD-ROM, dialing the global village, surfing the Internet, using multiple master, GX or Truetype fonts — all of these are only some of the buzz-words that we encounter on walls, in magazines, on our computer screens, and in the books we open. And what about Windows 95, NT, or other Unixes, can we just ignore them? No, we have to deal with them, adapt to the real world, profit from these developments, borrow the good ideas, use cross-fertilization to take what we need in order to make our tool of excellence — TEX — even better, and adapt it more ideally to the text processing needs of the year 2000. Recent developments such as Ω, ε−TEX, NTS, LATEX3, TDS, hyperTEX, ASTeR have shown that TEX is alive and well, and that many enthusiastic developers in various parts of the world are actively working on extensions in functionality to better integrate TEX into the window environments that are becoming commodity items in our daily life. TUG has to set up efficient communication channels, and the means to make coordinating all these activities possible. In particular, *TUGboat* will carry articles addressing these important issues so that everybody can be kept truly informed.

All together now

Many — I should say most — TEX users in the world are not members of TUG or of any other local or national TEX Users Group. Yet they all can profit from TEX's fantastic typesetting abilities. They are working in far-away places, on small personal computers. We should not forget those writing their thesis in Russian, research report in Chinese, perhaps a love letter in Armenian, or a poem in Swahili. TUG *per se* is not the prime aim of the game, it is not organizations that make history, it is people. Knuth gave TEX to the world, and asked TUG to look after it, to make sure that TEX and METAFONT can be used to the benefit of the whole of mankind as the only truly generally and freely available text-processing system in the world. Therefore we should all continue to work together, in trust and good faith. TUG depends on you, TUG needs your active support, and all those hundreds of thousands of TEX users depend on all of us. Let us not disappoint them!

◇ Christina Thiele
15 Wiltshire Circle
Nepean, Ontario
K2J 4K9, Canada
cthiele@ccs.carleton.ca

◇ Michel Goossens
CERN, CN Division
CH-1211 Geneva 23
Switzerland
goossens@cern.ch

Editorial Comments

Barbara Beeton

Apology to our readers

It will not have escaped your notice that this issue is late, very late. I won't bore you with the details, only point out that there have been some changes in circumstances that have made it impossible to spend the amount of time necessary to keep *TUGboat* on schedule by myself. However, help has been found, and a new production location that, unlike the AMS computer facilities, is accessible to other authorized members of the new production team. This facility, at the Supercomputer Computations Research Institute (SCRI) of Florida State University in Tallahassee, is due to the efforts of Mimi Burbank, and we are deeply appreciative of her help and that of the SCRI support staff.

An all-out effort has been undertaken to return *TUGboat* to schedule by the end of the year. This has already been described by the "old" and "new" presidents in their "Opening words". I'll introduce the new production team below.

Old faces, new faces

As with any volunteer-based endeavour, people who have given of their time and energy in one set of circumstances may find that they are unable to continue when circumstances change. This has happened once again on the *TUGboat* Editorial Board. Jackie Damrau, the first recipient of the Donald E. Knuth Scholarship, has been Associate Editor of the LaTeX column since 1986. Jackie has been a champion of the LaTeX user; her goal has been to find better ways of explaining "how to", as opposed to laying bare the gory details that are of interest mainly to a developer. Both are necessary, but we should not forget that this is a *users group*. We wish Jackie all the best in her future endeavours.

The new faces belong to the production team that has been drawn together for the task of putting *TUGboat* back on schedule. Mimi Burbank, mentioned above as our "angel", will assume the function of Production Manager. Christina Thiele, in addition to production duties, has agreed to take on the job of Associate Editor for Philology and Linguistics. Robin Fairbairns, Michel Goossens and Sebastian Rahtz complete the team. All these individuals have considerable experience in editing and producing TUG proceedings and/or other TeX user group publications as well as other TeX and computer-related talents too numerous to mention. They have come in and "hit the ground running" with an abundance of energy and good will. I remain the Editor-in-Chief, responsible for major decisions, overall guidance, and the person to complain to if something goes wrong. I'm confident that this team can deliver the goods.

MetaPost goes public

MetaPost, the extended METAFONT program by John Hobby that creates PostScript output for diagrams and similar non-alphabetic shapes, has been placed in the public domain by AT&T. MetaPost has facilities for including TeX output and manipulating the resulting picture.

The distribution can be retrieved by anonymous ftp from `netlib.att.com/netlib/...`. The user's manual and auxiliary documentation are in `.../att/cs/cstr` as `162.ps.Z` and `164.ps.Z` respectively. The source distribution is `.../research/metapost.tar.Z`. This is also mirrored at CTAN in `tex-archive/graphics/metapost`.

The distribution requires the Unix Web2C version of TeX and Tom Rokicki's `dvips`, which contains special support for using downloaded TeX fonts in included figures. A DOS version of MetaPost is promised.

◇ Barbara Beeton
American Mathematical Society
P. O. Box 6248
Providence, RI 02940 USA
bnb@math.ams.org

Software & Tools

Making MakeTeXPK safer for UNIX installations

Michał Jaegermann

Abstract

Leaving directories open for anybody in the world to write to, in order to ensure that `MakeTeXPK` will work, is often a security concern. This note describes how to avoid that on UNIX systems without losing functionality.

– – ∗ – –

If you run TEX on UNIX, or any other UNIX-like system, and you are using a very convenient and popular setup with automatic bitmap generation, then you'll undoubtedly notice that this requires directories with `'write'` permissions to everybody. In theory, this is not much different than having your `/tmp` directory open that way. In practice, though, especially when your programs are compiled with the `kpathsea` library and you have multiple unprotected font directories, this may cause substantial security and administrative headaches. An option to create automatically all bitmaps in one directory used only for that purpose, and to move them later by hand after careful examination to final locations, is not very practical if you serve multiple printers with different characteristics and is always unattractive to busy system administrators.

A first, but rather feeble line of defense, if your variant of UNIX supports it, is to set the 'sticky' bit on the directories in question like this: `chmod a`t pk+. That way files can be removed or overwritten only by their owner(s). Unfortunately this does not prevent a "denial of service" attack. For example, a perpetrator may fill up directories with garbage such as empty files with expected names thus preventing later generation of required bitmaps. The above is just a prank, without lasting long-term consequences, but repeated often enough it may turn into a major nuisance. There are other, more insidious possible threats, which do occur in practice, especially when an attacker comes over a network using compromised legitimate accounts.

If you ever even thought of running `MakeTeXPK` "suid" (that means, in a mode which gives a program all privileges of its *owner*) you should drop the idea immediately, especially if `MakeTeXPK` is owned by `root`. `MakeTeXPK` is a *shell script* and for many reasons running shell scripts in "suid" mode is one of the biggest security holes you can think of. Modern UNIX kernels usually simply disable "suid" scripts, but even if they are permitted on your system, avoid using them. Problems with "world writable" directories pale in comparison.

Fortunately, there are safer approaches. As long as you have a compiler and an editor, `MakeTeXPK` and relatives (`MakeTeXTFM`, `MakeTEXMF`, ...) may be executed indirectly by a small compiled "wrapper" program and this latter program can be safely made "suid".

The notes below describe the steps leading to such a modification. Due to assorted variations in TEX distributions and installations it is highly unlikely that they will work for you literally as given — unless you happen to run a very similar system. To make details easier to follow they will be shown for one particular distribution (`teTeX`, version 0.2, for Linux). This is only an example but it should give an idea how to proceed in other cases. Modifications should be rather straightforward. There are also possible variations in UNIX behaviour. See, for example, the Solaris note below.

It goes without saying that you need `root` access for all (okay, most) of these steps.

- Create new user account, say `tex`, on your system and include it in some innocuous group, e.g., `tex`.

The only purpose of this "user" will be to own common TEX files. You may already have some suitable group, such as `nogroup` or `nobody`, so instead you may include your new user there.

- Give the `tex` account `/bin/false` for a login shell and disable password by putting `*` in the corresponding field. Home directory is not terribly important. Nobody will ever be logging into this account.

The completed entry in the password file will look something like this:

```
tex:*:117:65535:Owner of TeX files:
                /usr/local/tex:/bin/false
```

- Do `touch /usr/spool/mail/tex` to create an empty file. This action closes a loophole in some mail delivery programs. You may find the file already in place, made by some system administration utility. Change owner and group of this file to those of root (`chown root:root /usr/spool/mail/tex`) and also remove all read and write permissions on it (`chmod 000 /usr/spool/mail/tex`).
- Make sure that `MakeTeXPK`, and similar scripts you want to execute the same way, are **not** in

your `$PATH`, or at least not earlier than the intended location of your "wrapper" program(s). Original scripts will not be called directly, however, the "wrapper" program will "inherit" their names thus presenting the same interface to users and other programs.

Any convenient location will do for scripts, but if your system conforms to the TEX Directory Structure, then a subdirectory of `$TEXMFROOT` will be a logical place. For the particular `teTeX` distribution, version 0.2, it is enough to delete links in a directory `/usr/local/bin`. Real scripts `MakeTeXPK`, `MakeTeXMF`, and `MakeTeXTFM` reside in `/usr/local/tex/scripts-0.2/bin/` and may be left there.

- Edit all scripts in question to supply absolute paths to all executables. Don't forget to perform this task in other scripts which may be called by our `MakeTEX...` script (`append_db` in the `teTeX` distribution). Change the mode used when creating new files to 444 and to 755 for directories.

A proper way to do this is to start a script with a series of shell variable definitions similar to

```
MF="/usr/local/bin/mf"
```

and replace all later occurrences of `mf` in the script by `$MF`. That way, if you later move your METAFONT executables to some other place, script editing will be limited to one place; similarly for other programs. Absolute locations are required since, for security reasons, we will limit `$PATH` only to `"/bin:/usr/bin"`. This means that in theory you may leave things like `test`, `echo` or `rm` alone. In the latter case, for "dangerous" commands like `rm -f`, it is still a good idea to replace them with definitions similar to `RM="/bin/rm -f"` to get better control of what you are really executing.

As a side effect this keeps user-aliased or redesigned versions of these commands from putting out unexpected and unwanted text or hanging because of a need for `tty` input as in the case of the common and usually desirable alias of `rm` to `rm -i`. The unplanned appearance of extraneous text on `stdout` is one of the most common reasons for the `MakeTeXPK` to fail on its first pass.

Depending on your level of mistrust, you may use a similar approach to `echo` and `test` as well, but in the sample files, I have chosen to be more relaxed. Moreover, they may be "built-ins" in your shell.

- Edit sample source given in the appendix to adjust it to your system and compile.
- Install results of the compilation somewhere in your `$PATH`. The directory `/usr/local/bin` is usually a good place. Go there and name your program `MakeTeXPK`. Change its ownership and group to that of user `tex` by typing `chmod tex:tex MakeTeXPK`. (Depending on your variant of UNIX you may have to use a dot instead of a colon to separate the user and the group name, or you may have to do that in two steps, using also another command, `chgrp`, to accomplish the above. Use the group to which you assigned your `tex` user. This is only an example). The program needs "execute" and "set uid" privileges (`chmod 4755 MakeTeXPK`). Also for your other `MakeTeX...` scripts, provide corresponding "call points" with their names via file links (`ln MakeTeXPK MakeTeXTFM`; `ln MakeTeXPK MakeTeXMF`).

Solaris note: Passing ownership privileges to a subprocess, as illustrated above, works for Linux and other assorted UNIX systems. Still, I am informed by Ulrik Vieth (`vieth@thphy.uni-duesseldorf.de`), that on Solaris systems this happens only when the wrapper is "suid" and owned by `root`. Therefore a

```
setuid(geteuid());
```

line from a sample source will not work as intended (either the call will fail or the subprocess will not be owned by the `tex` account). One should instead set explicitly `TEXGROUP` and `TEXUSER` of a type `uid_t` and include a replacement code like this

```
setgid(TEXGROUP);
setuid(TEXUSER);
```

in the given order — to achieve the same effect. This may apply as well to other UNIX variants. *Caveat emptor!*

- Change ownership of all your font files to `tex`. Actually you may make `tex` an owner of whole directory trees in TEX system files. Assuming that all you want to assign that way is in a tree rooted in `texmf`, you may accomplish that by doing `chown -R tex:tex texmf`. If your `chown` does not understand the `-R` (recursive) flag, then something similar to the following should do:

```
find texmf -print | xargs -n1 chown tex:tex
```

See also `chown` remarks in the previous item.

- Remove "write" permissions for anybody but owner on all directories in question. A command like the following one should accomplish that task (be careful, you do not want to change non-directories):

```
find texmf -type d -print | xargs -n1 chmod 755
```

You are done. Now, whenever `MakeTeXPK` is called directly from a command line or by some other program like `dvips`, your "wrapper" program will be executed instead. It will call, in turn, a "real" script but one already with the id of the owner of your font directories.

Concluding remarks

The presented solution is not entirely without problems. Due to "out of sync" ownership and permissions, `kpathsea` library functions may fail, depending on the exact moment this happened, when trying to write the `missfont.log` file in cases when font-making was not successful. This can likely be hard to resolve without modifying the library itself. If you do encounter this problem then a simple workaround would be to create an empty `missfont.log`, owned by you, and to give it write permissions for everybody (`touch missfont.log; chmod a+w missfont.log`). When you are done simply change permissions back to the original state.

Another possible trouble spot will occur when you have "private" fonts because you are conducting some font-making experiments, for example, and you would like to have bitmaps created in places owned by you and not by `tex` (otherwise you will not be able to remove the results of failed tests). In this case make yourself a private, executable copy of the `MakeTeXPK` script, edit it accordingly and make sure that it can be found *earlier* in your `$PATH` than the system program with the same name. You will not able to deposit anything in system directories, but this is most likely what you want anyway. Your script does not have to run "suid", so repeating all of the above is not necessary.

Last but not least, there is another possible approach to the whole problem. There are only a few commands (`mv`, `mkdir`, `chmod`) which have to modify TEX "system" directories. Instead of running the whole `MakeTeX` in "suid" mode you may write special versions (`texmv`, `texmkdir`, `texchmod`) of these, which would operate "suid" `tex`, and use them as replacements in the `MakeTeXPK` script whenever needed. Whether this is a better idea depends entirely on your situation and security requirements.

◇ Michał Jaegermann
10923 36 Avenue
Edmonton, Alberta,
Canada T6J 0B7
Email: michal@ellpspace.math.ualberta.ca,\\ michal@gortel.phys.ualberta.ca

Appendix – Sample wrapper program code

Sample C code for a wrapper program for `teTeX`, version 0.2, Linux distribution. Adapt with caution!

```
/*******************************************/
/*                                         */
/*  Executable wrapper for MakeTeX...      */
/*  programs.  Calls its namesake from     */
/*  TOOLS directory.  Provide links with   */
/*  different names to make it multipurpose*/
/*                                         */
/*  Michal Jaegermann, Feb 11 1995         */
/*                                         */
/*******************************************/

#include <unistd.h>
#include <string.h>
#include <stdlib.h>

#define VERSION_S "0.2"
/*
 * If you do not have an ANSI compiler you may
 * use an "explicit" single string in TOOLS
 * define; this is just a way to make future
 * modifications easier.
 */
#define TOOLS  "/usr/local/tex/scripts-" \
               VERSION_S "/bin/"
#define ASIZE  120

/*
 * This is a list of names under which we are
 * willing to execute. It must be NULL
 * terminated.
 */
const char *accepted[] = {
    "MakeTeXPK",
    "MakeTeXTFM",
    "MakeTeXMF",
    NULL
};

int
main(int argc, char **argv)
{
    char doer[ASIZE] = TOOLS;
    int idx = 0;
    /*
     * If your compiler is broken and the
     * construction below does not work then
     * "tail = strchr(doer, '\0');", or
     * equivalent, will serve as well.
     */
    char *tail = doer + (sizeof(TOOLS) - 1);
    char *start;

    /* find our base name */
    start = (start = strrchr(argv[0], '/')) ?
                        (start + 1) : argv[0];
    /* check if we are on the list */
    while (1) {
        if (NULL == accepted[idx])
            exit(1);      /* not on the list */
```

```
        if (0 == strcmp(accepted[idx], start))
            break;        /*  this is ours   */
        idx += 1;
    }
    /*
     * Set pretty bland, but hopefuly secure
     * environment; we intend to run this
     * program 'suid'.
     */
    setenv("PATH", "/bin:/usr/bin", 1);
    setenv("IFS", " ", 1);
    /*
     * You may want/need some other calls to
     * setenv().  For example, if your system
     * has an environment variable pointing to
     * shared libraries it should be set here.
     */

    /*
     * Attach our name at the end of a
     * directory string.  This assumes that
     * real scripts in TOOLS directory  will
     * be called by their own names (but
     * indirectly)
     */
    strcpy(tail, start);

    /*
     * Get the privileges of the owner of this
     * program, then execute the script and
     * return its results
     */
    setuid(geteuid());
    return execv(doer, argv);
}
```

Hyphenation Exception Log

Barbara Beeton

This is the periodic update of the list of words that TEX fails to hyphenate properly. The list last appeared in *TUGboat* 13, no. 4, starting on page 452. Everything listed there is repeated here. Owing to the length of the list, it has been subdivided into two parts: English words, and names and non-English words that occur in English texts.

This list is specific to the hyphenation patterns that appear in the original `hyphen.tex`, that is, the patterns for U.S. English. If such information for other patterns becomes available, consideration will be given to listing that too. (See below, "Hyphenation for languages other than English".)

In the list below, the first column gives results from TEX's `\showhyphens{...}`; entries in the second column are suitable for inclusion in a `\hyphenation{...}` list.

In most instances, inflected forms are not shown for nouns and verbs; note that all forms must be specified in a `\hyphenation{...}` list if they occur in your document.

Thanks to all who have submitted entries to the list. Since some suggestions demonstrated a lack of familiarity with the rules of the hyphenation algorithm, here is a short reminder of the relevant idiosyncrasies. Hyphens will not be inserted before the number of letters specified by `\lefthyphenmin`, nor after the number of letters specified by `\righthyphenmin`. For U.S. English, `\lefthyphenmin=2` and `\righthyphenmin=3`; thus no word shorter than five letters will be hyphenated. (For the details, see *The TEXbook*, page 454. For a digression on other views of hyphenation rules, see below under "English hyphenation".) This particular rule is violated in some of the words listed; however, if a word is hyphenated correctly by TEX except for "missing" hyphens at the beginning or end, it has not been included here.

Some other permissible hyphens have been omitted for reasons of style or clarity. While this is at least partly a matter of personal taste, an author should think of the reader when deciding whether or not to permit just one more break-point in some obscure or confusing word. There really are times when a bit of rewriting is preferable.

One other warning: Some words can be more than one part of speech, depending on context, and have different hyphenations; for example, 'analyses' can be either a verb or a plural noun. If such a word appears in this list, hyphens are shown only for the portions of the word that would be hyphenated the same regardless of usage. These words are marked with a '*'; additional hyphenation points, if needed in your document, should be inserted with discretionary hyphens.

Words added since the last appearance of the list are preceded by $^+$.

The reference used to check these hyphenations is *Webster's Third New International Dictionary*, Unabridged.

English hyphenation

It has been pointed out to me that the hyphenation rules of British English are based on the etymology of the words being hyphenated as opposed to the "syllabic" principles used in the U.S. Furthermore, in the U.K., it is considered bad style to hyphenate a word after only two letters. In order to make TEX

defer hyphenation until after three initial letters, set `\lefthyphenmin=3`.

Of course, British hyphenation patterns should be used as well. A set of patterns for UK English has been created by Dominik Wujastyk and Graham Toal, using Frank Liang's `PATGEN` and based on a file of 114925 British-hyphenated words generously made available to Dominik by Oxford University Press. (This list of words and the hyphenation break points in the words are copyright to the OUP and may not be redistributed.) The file of hyphenation patterns may be freely distributed; it is posted on CTAN in the file

`tex-archive/language/english/ukhyph.tex`

and can be retrieved by anonymous FTP.

Hyphenation for languages other than English

Patterns now exist for many languages other than English, including languages using accented alphabets. CTAN holds an extensive collection of patterns in subdirectories of

`tex-archive/language`

The List — English words

`academy(ies)`	acad-e-my(ies)
`addable`	add-a-ble
`ad-di-ble`	add-i-ble
`adrenaline`	adren-a-line
+ `aerospace`	aero-space
`af-terthought`	af-ter-thought
`agronomist`	agron-o-mist
`am-phetamine`	am-phet-a-mine
`anal-yse(d)`	an-a-lyse(d)
`anal-y-ses`	analy-ses *
`anomaly(ies)`	anom-aly(ies)
`an-tideriva-tive`	an-ti-deriv-a-tive
`anti-nomy(ies)`	an-tin-o-my(ies)
`antin-u-clear`	an-ti-nu-clear
`antin-u-cleon`	an-ti-nu-cle-on
`an-tirev-o-lu-tion-ary`	an-ti-rev-o-lu-tion-ary
`apotheoses`	apoth-e-o-ses
`apotheo-sis`	apoth-e-o-sis
`ap-pendix`	ap-pen-dix
`archipelago`	arch-i-pel-ago
`archety-pal`	ar-che-typ-al
`archetyp-i-cal`	ar-che-typ-i-cal
`arc-t-an-gent`	arc-tan-gent (better: arc tangent)
`assignable`	as-sign-a-ble
`as-sig-nor`	as-sign-or
`as-sis-tantship`	as-sist-ant-ship
`asymp-tomatic`	asymp-to-matic
`asymp-totic`	as-ymp-tot-ic
`asyn-chronous`	asyn-chro-nous
`atheroscle-ro-sis`	ath-er-o-scle-ro-sis
`at-mo-sphere`	at-mos-phere
`at-tributed`	at-trib-uted
`at-tributable`	at-trib-ut-able
`avoirdupois`	av-oir-du-pois
`awo-ken`	awok-en
`ban-dleader`	band-leader
`bankrupt(cy)`	bank-rupt(-cy)
`ba-ronies`	bar-onies
`base-li-neskip`	`\base-line-skip`
`bathymetry`	ba-thym-e-try
`bathyscaphe`	bathy-scaphe
`bea-nies`	bean-ies
`be-haviour`	be-hav-iour
`be-vies`	bevies
+ `bib-li-og-ra-physstyle`	`\bib-li-og-ra-phy-style`
`bid-if-fer-en-tial`	bi-dif-fer-en-tial
+ `biggest`	big-gest
`bil-l-able`	bill-able
`biomath-e-mat-ics`	bio-math-e-mat-ics
`biomedicine`	bio-med-i-cine
`biorhythms`	bio-rhythms
`bitmap`	bit-map
`blan-der`	bland-er
`blan-d-est`	bland-est
`blin-der`	blind-er
`blon-des`	blondes
`blueprint`	blue-print
`bornolog-i-cal`	bor-no-log-i-cal
`bo-tulism`	bot-u-lism
`brus-quer`	brusquer
`bus-ier`	busier
`bus-i-est`	busiest
`buss-ing`	bussing
`but-ted`	butted
`buz-zword`	buzz-word
`ca-caphony(ies)`	ca-caph-o-ny(ies)
`cam-er-a-men`	cam-era-men
`cartwheel`	cart-wheel
`catar-rhs`	ca-tarrhs
`catas-trophic`	cat-a-stroph-ic
`catas-troph-i-cally`	cat-a-stroph-i-cally
`cauliflower`	cau-li-flow-er
`cha-parral`	chap-ar-ral
`chartreuse`	char-treuse
`cholesteric`	cho-les-teric
`cigarette`	cig-a-rette
`cin-que-foil`	cinque-foil
`cognac`	co-gnac
`cog-nacs`	co-gnacs
`com-parand`	com-par-and
`com-para-nds`	com-par-ands
`comptroller`	comp-trol-ler
`con-formable`	con-form-able
`con-formist`	con-form-ist
`con-for-mity`	con-form-ity
`congress`	con-gress
+ `con-tribute(s,d)`	con-trib-ute(s,d)
`cose-cant`	co-se-cant
`cotan-gent`	co-tan-gent
+ `courses`	cour-ses
`crankshaft`	crank-shaft
`crocodile`	croc-o-dile
`crosshatch(ed)`	cross-hatch(ed)
`dachshund`	dachs-hund
`database`	data-base

dat-a-p-ath	data-path
declarable	de-clar-able
defini-tive	de-fin-i-tive
delectable	de-lec-ta-ble
democratism	de-moc-ra-tism
de-mos	demos
deriva-tive	de-riv-a-tive
diffract	dif-fract
di-rer	direr
di-re-ness	dire-ness
dis-parand	dis-par-and
dis-traugh-tly	dis-traught-ly
dis-tribute(d)	dis-trib-ute(d)
dou-blespace(ing)	dou-ble-space(-ing)
dol-lish	doll-ish
drif-tage	drift-age
driver(s)	dri-ver(s)
dromedary(ies)	drom-e-dary(ies)
duopolist	du-op-o-list
duopoly	du-op-oly
eas-t-en-ders	east-end-ers
eco-nomics	eco-nom-ics
economist	econ-o-mist
elec-trome-chan-i-cal	electro-mechan-i-cal
elec-tromechanoa-cous-tic	electro-mechano-acoustic
eli-tist	elit-ist
en-trepreneur(ial)	en-tre-pre-neur(-ial)
epinephrine	ep-i-neph-rine
equiv-ari-ant	equi-vari-ant
ethy-lene	eth-yl-ene
ev-ersible	ever-si-ble
ev-ert(s,ed,ing)	evert(s,-ed,-ing)
exquisite	ex-quis-ite
ex-traor-di-nary	ex-tra-or-di-nary
fermions	fermi-ons
flag-el-lum(la)	fla-gel-lum(-la)
flammables	flam-ma-bles
fledgling	fledg-ling
flowchart	flow-chart
formidable(y)	for-mi-da-ble(y)
forsythia	for-syth-ia
forthright	forth-right
freeloader	free-loader
friendlier	friend-lier
frivolity	fri-vol-ity
frivolous	friv-o-lous
+ galac-tic	ga-lac-tic
+ galaxy(ies)	gal-axy(-ies)
ga-some-ter	gas-om-e-ter
geodesic	ge-o-des-ic
geode-tic	ge-o-det-ic
ge-o-met-ric	geo-met-ric
geotropism	ge-ot-ro-pism
gnomon	gno-mon
grievance	griev-ance
grievous(ly)	griev-ous(-ly)
hairstyle	hair-style
hairstylist	hair-styl-ist
harbinger	har-bin-ger
harlequin	har-le-quin
hatcheries	hatch-eries
hemoglobin	he-mo-glo-bin
hemophilia	he-mo-phil-ia
+ hemorhe-ol-ogy	hemo-rhe-ol-ogy
hep-atic	he-pat-ic
hermaphrodite(ic)	her-maph-ro-dite(-ic)
heroes	he-roes
hex-adec-i-mal	hexa-dec-i-mal
holon-omy	ho-lo-no-my
ho-mo-th-etic	ho-mo-thetic
horseradish	horse-rad-ish
hy-potha-la-mus	hy-po-thal-a-mus
ide-als	ideals
ideographs	ideo-graphs
id-iosyn-crasy	idio-syn-crasy
ig-niter	ig-nit-er
ig-n-i-tor	ig-ni-tor
ig-nores-paces	\ignore-spaces
impedances	im-ped-ances
in-finitely	in-fin-ite-ly
in-finites-i-mal	in-fin-i-tes-i-mal
in-fras-truc-ture	in-fra-struc-ture
in-ter-dis-ci-plinary	in-ter-dis-ci-pli-nary
+ in-ter-galac-tic	in-ter-ga-lac-tic
inu-tile	in-utile
inu-til-ity	in-util-i-ty
ir-re-vo-ca-ble	ir-rev-o-ca-ble
itinerary(ies)	itin-er-ary(-ies)
jeremi-ads	je-re-mi-ads
keystroke	key-stroke
kil-ning	kiln-ing
la-ciest	lac-i-est
lamentable	lam-en-ta-ble
land-sca-per	land-scap-er
larceny(ist)	lar-ce-ny(-ist)
+ let-terspac-ing	let-ter-spac-ing
lifes-pan	life-span
lightweight	light-weight
limousines	lim-ou-sines
linebacker	line-backer
lines-pac-ing	\line-spacing
lithographed	lith-o-graphed
lithographs	lith-o-graphs
lobotomy(ize)	lo-bot-omy(-ize)
lo-ges	loges
+ longest	long-est
macroe-co-nomics	macro-eco-nomics
malapropism	mal-a-prop-ism
manuscript	man-u-script
marginal	mar-gin-al
+ math-e-mati-cian	math-e-ma-ti-cian
mat-tes	mattes
med-i-caid	med-ic-aid
mediocre	medi-ocre
medi-o-crities	medi-oc-ri-ties
me-galith	mega-lith
metabolic	meta-bol-ic
metabolism	me-tab-o-lism
met-a-lan-guage	meta-lan-guage
metropo-lis(es)	me-trop-o-lis(es)
metropoli-tan	met-ro-pol-i-tan
mi-croe-co-nomics	micro-eco-nomics
mi-crofiche	mi-cro-fiche
mil-lage	mill-age
milliliter	mil-li-liter
mimeographed	mimeo-graphed
mimeographs	mimeo-graphs
mimi-cries	mim-ic-ries

mi-nis	min-is
+ min-isym-po-sium(a)	mini-sym-po-sium(a)
min-uter(est)	mi-nut-er(-est)
mis-chievously	mis-chie-vous-ly
mis-ers	mi-sers
mis-ogamy	mi-sog-a-my
mod-elling	mod-el-ling
molecule	mol-e-cule
monar-chs	mon-archs
mon-eylen-der	money-len-der
monochrome	mono-chrome
mo-noen-er-getic	mono-en-er-getic
monoid	mon-oid
monopole	mono-pole
monopoly	mo-nop-oly
monos-pline	mono-spline
monos-trofic	mono-strofic
mono-tonies	mo-not-o-nies
monotonous	mo-not-o-nous
mo-ro-nism	mo-ron-ism
mosquito	mos-qui-to
mu-d-room	mud-room
mul-ti-faceted	mul-ti-fac-eted
mul-ti-pli-ca-ble	mul-ti-plic-able
mul-tiuser	multi-user (better with explicit hyphen)
ne-ofields	neo-fields
+ neon-azi	neo-nazi
newslet-ter	news-let-ter
non-ame	no-name
none-mer-gency	non-emer-gency
nonequiv-ari-ance	non-equi-vari-ance
noneu-clidean	non-euclid-ean
non-i-so-mor-phic	non-iso-mor-phic
nonpseu-do-com-pact	non-pseudo-com-pact
non-s-mooth	non-smooth
nonuni-form(ly)	non-uni-form(-ly)
nore-pinephrine	nor-ep-i-neph-rine
nutcracker	nut-crack-er
oer-st-eds	oer-steds
oligopolist	oli-gop-o-list
oligopoly(ies)	oli-gop-oly(ies)
operand(s)	op-er-and(s)
orangutan	orang-utan
or-thodon-tist	or-tho-don-tist
or-thok-er-a-tol-ogy	or-tho-ker-a-tol-ogy
or-thoni-tro-toluene	ortho-nitro-toluene (or-tho-ni-tro-tol-u-ene)
overview	over-view
ox-i-dic	ox-id-ic
+ padding	pad-ding
painlessly	pain-less-ly
pal-mate	palmate
parabola	par-a-bola
parabolic	par-a-bol-ic
paraboloid	pa-rab-o-loid
paradigm	par-a-digm
parachute	para-chute
paradimethyl-ben-zene	para-di-methyl-benzene (para-di-meth-yl-ben-zene)
paraflu-o-ro-toluene	para-fluoro-toluene (para-flu-o-ro-tol-u-ene)
para-g-ra-pher	para-graph-er
par-ale-gal	para-le-gal
par-al-lelism	par-al-lel-ism
para-m-ag-netism	para-mag-net-ism
paramedic	para-medic
param-ethy-lanisole	para-methyl-anisole (para-meth-yl-an-is-ole)
parametrize	pa-ram-e-trize
paramil-i-tary	para-mil-i-tary
paramount	para-mount
pathogenic	path-o-gen-ic
pee-vish(ness)	peev-ish(-ness)
pen-tagon	pen-ta-gon
petroleum	pe-tro-le-um
phe-nomenon	phe-nom-e-non
philatelist	phi-lat-e-list
phos-pho-ric	phos-phor-ic
pi-cador	pic-a-dor
pi-ran-has	pi-ra-nhas
pla-ca-ble	placa-ble
plea-sance	pleas-ance
poltergeist	pol-ter-geist
polyene	poly-ene
polyethy-lene	poly-eth-yl-ene
polygamist(s)	po-lyg-a-mist(s)
poly-go-niza-tion	polyg-on-i-za-tion
polyphonous	po-lyph-o-nous
polystyrene	poly-styrene
pomegranate	pome-gran-ate
poroe-las-tic	poro-elas-tic
+ porous	por-ous
postam-ble	post-am-ble
postscript	post-script
pos-tu-ral	pos-tur-al
pream-ble	pre-am-ble
preloaded	pre-loaded
prepar-ing	pre-par-ing
+ preprint(s)	pre-print(s)
pre-pro-ces-sor	pre-proces-sor
pre-s-plit-ting	\pre-split-ting
priestesses	priest-esses
+ pret-typrinter	pret-ty-prin-ter
pro-ce-du-ral	pro-ce-dur-al
pro-cess	process*
procu-rance	pro-cur-ance
pro-ge-nies	prog-e-nies
progeny	prog-e-ny
pro-hibitive(ly)	pro-hib-i-tive(-ly)
prosci-utto	pro-sciut-to
protester(s)	pro-test-er(s)
protestor(s)	pro-tes-tor(s)
pro-to-ty-pal	pro-to-typ-al
pseu-dod-if-fer-en-tial	pseu-do-dif-fer-en-tial
pseud-ofi-nite	pseu-do-fi-nite
pseud-ofinitely	pseu-do-fi-nite-ly
pseud-o-forces	pseu-do-forces
pseudonym	pseu-do-nym
pseu-doword	pseu-do-word
psychedelic	psy-che-del-ic
psy-chs	psychs
pubescence	pu-bes-cence
+ quadding	quad-ding
quadratic(s)	qua-drat-ic(s)
quadra-ture	quad-ra-ture
quadriplegic	quad-ri-pleg-ic
quain-ter(est)	quaint-er(-est)

quasiequiv-a-lence	qua-si-equiv-a-lence or quasi-
quasi-hy-ponor-mal	qua-si-hy-po-nor-mal
quasir-ad-i-cal	qua-si-rad-i-cal
quasiresid-ual	qua-si-resid-ual
qua-sis-mooth	qua-si-smooth
qua-sis-ta-tion-ary	qua-si-sta-tion-ary
qu-a-sito-pos	qua-si-topos
qu-a-si-tri-an-gu-lar	qua-si-tri-an-gu-lar
quintessence	quin-tes-sence
quintessen-tial	quin-tes-sen-tial
rab-bi-try	rab-bit-ry
ra-dio-g-ra-phy	ra-di-og-ra-phy
raf-f-ish(ly)	raff-ish(-ly)
ramshackle	ram-shackle
ravenous	rav-en-ous
re-ar-range-ment	re-arrange-ment
re-ciproc-i-ties	rec-i-proc-i-ties
reci-procity	rec-i-proc-i-ty
rect-an-gle	rec-tan-gle
ree-cho	re-echo
+ reprint(s)	re-print(s)
restorable	re-stor-able
re-tri-bu-tions	ret-ri-bu-tions
retrofit(ted)	retro-fit(-ted)
rhinoceros	rhi-noc-er-os
righ-teous(ness)	right-eous(-ness)
ringleader	ring-leader
robot	ro-bot
robotics	ro-bot-ics
roundtable	round-table
salesclerk	sales-clerk
salescle-rks	sales-clerks
saleswoman(en)	sales-woman(en)
salmonella	sal-mo-nel-la
sarsaparilla	sar-sa-par-il-la
sauerkraut	sauer-kraut
sca-to-log-i-cal	scat-o-log-i-cal
schedul-ing	sched-ul-ing
schizophrenic	schiz-o-phrenic
schnauzer	schnau-zer
schoolchild(ren)	school-child(-ren)
schoolteacher	school-teacher
scy-thing	scyth-ing
+ sec-re-tariat	sec-re-tar-iat
semaphore	sem-a-phore
semester	se-mes-ter
semidef-i-nite	semi-def-i-nite
semi-ho-mo-th-etic	semi-ho-mo-thet-ic
semir-ing	semi-ring
semiskilled	semi-skilled
seroepi-demi-o-log-i-cal	sero-epi-de-mi-o-log-i-cal
ser-vomech-a-nism	ser-vo-mech-anism
setup	set-up
severely	se-vere-ly
sha-peable	shape-able
shoestring	shoe-string
sidestep	side-step
sideswipe	side-swipe
skyscraper	sky-scraper
smokestack	smoke-stack
snorke-l-ing	snor-kel-ing
solenoid	so-le-noid
so-lute(s)	solute(s)
sovereign	sov-er-eign
spaces	spa-ces
specious	spe-cious
spelunker	spe-lunk-er
spendthrift	spend-thrift
spheroid(al)	spher-oid(-al)
sph-inges	sphin-ges
spi-cily	spic-i-ly
spinors	spin-ors
spokeswoman(en)	spokes-woman(en)
sportscast	sports-cast
sportively	spor-tive-ly
sportswear	sports-wear
sportswriter	sports-writer
sprightlier	spright-lier
squeamish	squea-mish
stan-dalone	stand-alone
startling(ly)	star-tling(-ly)
statis-tics	sta-tis-tics
stealthily	stealth-ily
steeplechase	steeple-chase
stochas-tic	sto-chas-tic
strangeness	strange-ness
stratagem	strat-a-gem
stretchier	stretch-i-er
stronghold	strong-hold
+ strongest	strong-est
stupi-der(est)	stu-pid-er(est)
summable	sum-ma-ble
su-perego	super-ego
su-pere-gos	super-egos
supremacist	su-prema-cist
surveil-lance	sur-veil-lance
swim-m-ingly	swim-ming-ly
symp-tomatic	symp-to-matic
syn-chromesh	syn-chro-mesh
syn-chronous	syn-chro-nous
syn-chrotron	syn-chro-tron
talkative	talk-a-tive
tapestry(ies)	ta-pes-try(ies)
tarpaulin	tar-pau-lin
tele-g-ra-pher	te-leg-ra-pher
telekinetic	tele-ki-net-ic
teler-obotics	tele-ro-bot-ics
+ tenure	ten-ure
testbed	test-bed
+ tex-twidth	text-width
tha-la-mus	thal-a-mus
ther-moe-las-tic	ther-mo-elas-tic
times-tamp	time-stamp
toolkit	tool-kit
to-po-graph-i-cal	topo-graph-i-cal
to-ques	toques
traitorous	trai-tor-ous
transceiver	trans-ceiver
transgress	trans-gress
transver-sal(s)	trans-ver-sal(s)
transvestite	trans-ves-tite
traversable	tra-vers-a-ble
traver-sal(s)	tra-ver-sal(s)
treacheries	treach-eries
troubadour	trou-ba-dour
+ turkey	tur-key
turnaround	turn-around

ty-pal	typ-al
unattached	un-at-tached
unerringly	un-err-ing-ly
un-friendly(ier)	un-friend-ly(i-er)
va-guer	vaguer
vaudeville	vaude-ville
vi-cars	vic-ars
vil-lai-ness	vil-lain-ess
viviparous	vi-vip-a-rous
voiceprint	voice-print
vs-pace	\vspace
+ wadding	wad-ding
wallflower	wall-flower
wastew-a-ter	waste-water
waveg-uide	wave-guide
+ wavelets	wave-lets
we-b-like	web-like
weeknight	week-night
wheelchair	wheel-chair
whichever	which-ever
whitesided	white-sided
whites-pace	white-space
widespread	wide-spread
wingspread	wing-spread
witchcraft	witch-craft
+ wordspac-ing	word-spac-ing
workhorse	work-horse
wraparound	wrap-around
wretched(ly)	wretch-ed(-ly)
yesteryear	yes-ter-year

Names and non-English words used in English text

al-ge-brais-che	al-ge-brai-sche
Al-legheny	Al-le-ghe-ny
Arkansas	Ar-kan-sas
+ Aus-tralasian	Aus-tral-asian
au-toma-tisierter	auto-mati-sier-ter
Be-di-enung	Be-die-nung
bib-li-ographis-che	bib-li-o-gra-phi-sche
+ Boston	Bos-ton
Brow-n-ian	Brown-ian
+ Brunswick	Bruns-wick
+ Bu-dapest	Bu-da-pest
+ Caribbean	Car-ib-bean
+ Charleston	Charles-ton
+ Char-lottesville	Char-lottes-ville
Columbia	Co-lum-bia
Czechoslo-vakia	Czecho-slo-va-kia
Di-jk-stra	Dijk-stra
dy-namis-che	dy-na-mi-sche
En-glish	Eng-lish
Eu-le-rian	Euler-ian
+ Evanston	Evan-ston
Febru-ary	Feb-ru-ary
Festschrift	Fest-schrift
Florida	Flor-i-da
Forschungsin-sti-tut	For-schungs-in-sti-tut
funk-t-sional	funk-tsional
Gaus-sian	Gauss-ian
Greif-swald	Greifs-wald
Grothendieck	Grothen-dieck
Grundlehren	Grund-leh-ren
+ Haifa	Hai-fa
Hamil-to-nian	Hamil-ton-ian
+ Helsinki	Hel-sinki
Her-mi-tian	Her-mit-ian
Hi-bbs	Hibbs
+ Hokkaido	Hok-kai-do
Jan-uary	Jan-u-ary
Japanese	Japan-ese
Kadomt-sev	Kad-om-tsev
Kansas	Kan-sas
Karl-sruhe	Karls-ruhe
Ko-rteweg	Kor-te-weg
+ Lan-caster	Lan-cas-ter
Leg-en-dre	Le-gendre
Le-ices-ter	Leices-ter
Lip-s-chitz(ian)	Lip-schitz(-ian)
Louisiana	Lou-i-si-ana
Manch-ester	Man-ches-ter
Marko-vian	Mar-kov-ian
Mas-sachusetts	Mass-a-chu-setts
+ Min-neapo-lis	Min-ne-ap-o-lis
Min-nesota	Min-ne-sota
+ Moscow	Mos-cow
+ Nachrichten	Nach-richten
+ Nashville	Nash-ville
Ni-jmegen	Nij-me-gen
Noethe-rian	Noe-ther-ian
No-ord-wi-jk-er-hout	Noord-wijker-hout
Novem-ber	No-vem-ber
+ Palermo	Pa-ler-mo
+ Philadel-phia	Phil-a-del-phia
Poincare	Poin-care
Po-ten-tial-gle-ichung	Po-ten-tial-glei-chung
rathskeller	raths-kel-ler
Rie-man-nian	Rie-mann-ian
Ry-d-berg	Ryd-berg
schot-tis-che	schot-tische
Schrodinger	Schro-ding-er
Schwabacher	Schwa-ba-cher
Schwarzschild	Schwarz-schild
Septem-ber	Sep-tem-ber
Stokess-che	Stokes-sche
+ Stuttgart	Stutt-gart
Susque-hanna	Sus-que-han-na
tech-nis-che	tech-ni-sche
Ten-nessee	Ten-nes-see
+ Ukrainian	Ukrain-ian
ve-r-all-ge-mein-erte	ver-all-ge-mein-erte
+ Vere-ini-gung	Ver-ein-i-gung
Verteilun-gen	Ver-tei-lun-gen
Wahrschein-lichkeit-s-the-o-rie	Wahr-schein-lich-keits-theo-rie
Werthe-rian	Wer-ther-ian
Winch-ester	Win-ches-ter
Yingy-ong Shuxue Jisuan	Ying-yong Shu-xue Ji-suan
+ Zealand	Zea-land
Zeitschrift	Zeit-schrift

Philology

Configuring TEX or LATEX for typesetting in several languages

Claudio Beccari

Abstract

Based on the frequent requests for help that I have received in the past months from users who want to configure TEX or LATEX for typesetting in several languages, it seems that this sort of information is not adequately covered in the documentation available to most do-it-yourself users. This tutorial is intended to give basic information on this topic. The related subject of hyphenation patterns will be addressed in a future tutorial.

1 Introduction

LATEX users are excused for not knowing how to set up their favorite typesetting program for different languages; Lamport [5] doesn't say a word on this subject; the new edition invites the reader to consult Goossens et al. [3] for what concerns multilanguage typesetting, and certainly the latter contains several hints — more than simple hints, since all of chapter 9 is dedicated to this problem.

TEX users should be more familiar with this problem via the main reference, Knuth [4], which covers the topic of hyphenation patterns. Appendix H stresses that TEX hyphenation capabilities are generated by the `\pattern` and the `\hyphenation` commands, the former being processable only by the initialization TEX program `initex`.

J. Braams prepared a complete system of style files and macros for use with LATEX, called babel; officially, it works only with standard document styles, but in practice, it is also valid for other styles and includes adequate means for setting text into type in some twenty languages. The only part that is lacking from the babel system is the set of hyphenation patterns for each of those languages, but this was done on purpose, I suppose, because pattern preparation, although essential for multilanguage typesetting, has almost nothing to do with the *style* of multilanguage typesetting.

This tutorial will attempt to explain to do-it-yourself (LA)TEX users how to configure their systems in order to set text into type in different languages at the same time. It is natural that this issue should be particularly interesting for non-English-speaking (LA)TEX users, but I have received requests for help from the United States as well, so I presume that across the ocean there are also some people who may benefit from these simple notes.

Typesetting text in several languages implies the following problems: (a) correct hyphenation, (b) correct "labels" in titles and captions ("Chapter", "Capitolo", "Chapitre", etc.), (c) special language-dependent typesetting rules (French vs. non-French spacing, quotation marks, etc.). Therefore the points to be covered include:

1. choosing the proper fonts and font encodings;
2. retrieving or creating hyphenation patterns;
3. initializing TEX or LATEX, taking into account the program's memory limitations;
4. retrieving or creating macros for switching languages;
5. language-dependent typesetting macros.

Here I will skip over the problem of typesetting with alphabets different from the extended Latin one; for Greek (modern and ancient) there is plenty of information (TEX and METAFONT files, the latter for generating the full set of characters and ligatures) in the CTAN directories

```
/tex-archive/languages/greek/levy
/tex-archive/languages/greek/yannis
```

The former contains the full set of 256-glyph fonts and TEX macros for handling them, the latter contains also the 128-glyph fonts, TEX macros and hyphenation patterns; both contain excellent documentation for using Greek fonts and for setting ancient and modern Greek.

The platforms (or boxes, as TEXies seem to call their computers) on which people set their texts are of widely different types, with different operating systems and, in particular, with different file systems. I'll address myself mainly to the DOS type of personal computer users, because apparently this category counts the highest number of do-it-yourself users; with minor modifications what I'll say is applicable also to other environments, with the caution that with VMS one must have system manager privileges and with UNIX, superuser privileges. In any case I assume that the do-it-yourself reader who is going to use this information in practice is sufficiently familiar with his/her computer to be able to use the available utilities for exploring and managing the hard disk(s) and the file system, for creating command (or batch or script) files, and so on.

As for myself, I use a VMS mainframe, a UNIX workstation and a DOS personal computer; the latter is a 486 type and has 8 Mb of RAM, so that, even though I don't use Windows, with a suitable

extended memory handler I can use a "big TEX" implementation of TEX based on the compilation of a C-source program; I recommend that any DOS user equip him/herself with a configuration of this sort. My LATEX is set up to deal with eight languages at a time: US-English, Italian, French, Spanish, Portuguese, Catalan, Romanian, and Latin (modern spelling). Except for US-English, I prepared all the hyphenation patterns for the remaining seven languages myself.

2 Fonts

Except for English (both US and UK), which does not use diacritics (actually it does when assimilated foreign words are used), and textual Latin (i.e. excluding prosodic Latin), almost all the languages that were examined by Sojka and Ševeček [6] use diacritics. (LA)TEX has no problem in setting suitable diacritics over or under any letter, but (LA)TEX has real problems in hyphenating words that contain the control symbols or control words that are used for setting such diacritics. In fact, TEX "words" are not the same as the words of a language. Loosely speaking, for TEX a "word" is a sequence of characters of category code 11 (letter) or 12 (other), with a non-zero `\lccode`, set in the same font, that is preceded by a punctuation mark (parenthesis, quotation marks, etc.) or by space or glue, and ends with anything different from the above-mentioned characters. Any command that interrupts this sequence interrupts the word, unless it expands to one or more characters with the proper characteristics. For a precise definition of what TEX thinks a word is, see Appendix H of *The TEXbook*.

As examples, the French word `\'ecole` = *école* is not a word at all for TEX because the accent command `\'` expands to `\accent 19`, unless ... Even the English word `{\it school}` = *school* is not a word for TEX, because there is no space or glue between the command `\it` and the word 'school'; this is why (LA)TEX often reports overfull hboxes when you emphasize some text.

Unless ... yes, there is a way around this: extended fonts and a simple little macro, `\hz`, for inserting glue where there should be no extra space:

```
\def\hz{\nobreak\hskip0pt \relax}
```

In fact, `\hz` inserts an unbreakable glob of glue of zero width, stretch and shrink, but nevertheless it is glue, so that after this glob a real TEX word may begin, a word that TEX can hyphenate properly. Use this little macro and you'll get rid of many hyphenation problems. For example, if you type

```
\emph{\hz electricity}
```

you get *electricity* properly hyphenated even though it is emphasized. If you want to typeset some French words within an English text, and you only have the standard cm fonts, besides inserting a lot of discretionary breaks `\-` at every syllable (if you don't have the French hyphenation patterns), you have the possibility of inserting `\hz` in order to convince TEX to deal with word fragments; for example, you can type

```
\'e\hz lectricit\'e
```

and TEX will try to hyphenate the fragment 'lectricit', probably finding some correct hyphen points even by using English patterns.

But the real solution lies in using the extended fonts. There are two flavors: the real dc fonts and the virtual em fonts. The former are complete sets of 256 glyphs that conform to the "double Cork" encoding, while the latter are virtual fonts that are made up with pieces taken from real 128-glyph cm fonts. Although em fonts lack some glyphs, compared with real dc fonts, they sometimes are more flexible than dc fonts since by editing the virtual property list file it is possible to modify them very easily. Moreover the `.pk` files for the dc fonts in the customary 300, 329, ..., 746 dots-per-inch sizes occupy approximately 7 Mb of disk space, which might not be available on the smaller platforms or which might be used in a different way.

If you already have the dc `.tfm` and pixel files on your computer disk, or if you are willing to copy them from a CTAN archive, skip the next section; otherwise, you might find it interesting to create the full set of em fonts yourself, as explained below.

2.1 Creating standard virtual em fonts

Examine your file system and find out if you already have `.tfm` files whose names start with `em` and if you have files with the same name but extension `.vf`; if you do, skip to the next section.

If you don't, you might be in trouble, but before giving up examine your file system and find out if you have executable files (extension `.exe`) with the following names: `tftovp`, `vftovp`, `vptovf`, `pltotf`, `tftopl`. If you do, they probably came with your screen and/or printer driver, unless you are the type of hacker who copies everything in the hope that it might become useful at some future date.

This point is crucial; in fact, if you got these files with your drivers, you can be almost certain that your drivers handle virtual fonts. Check the driver documentation; if your drivers actually handle virtual fonts, keep reading this section; otherwise, go to the next subsection. There is no sense

	’0	’1	’2	’3	’4	’5	’6	’7	
’20	Ă	Ą	Ć	Č	Ď	Ě	Ę	Ğ	"8
’21	Ĺ	L’	Ł	Ń	Ň		Ő	Ŕ	
’22	Ř	Ś	Š	Ş	Ť	Ţ	Ű	Ů	"9
’23	Ÿ	Ź	Ž	Ż	IJ	İ	đ	§	
’24	ă	ą	ć	č	d’	ě	ę	ğ	"A
’25	í	l’	ł	ń	ň		ő	ŕ	
’26	ř	ś	š	ş	t’	ţ	ű	ů	"B
’27	ÿ	ź	ž	ż	ij	¡	¿		
’30	À	Á	Â	Ã	Ä	Å	Æ	Ç	"C
’31	È	É	Ê	Ë	Ì	Í	Î	Ï	
’32		Ñ	Ò	Ó	Ô	Õ	Ö	Œ	"D
’33	Ø	Ù	Ú	Û	Ü	Ý		SS	
’34	à	á	â	ã	ä	å	æ	ç	"E
’35	è	é	ê	ë	ì	í	î	ï	
’36		ñ	ò	ó	ô	õ	ö	œ	"F
’37	ø	ù	ú	û	ü	ý		ß	
	"8	"9	"A	"B	"C	"D	"E	"F	

Table 1: Font layout for em fonts with character codes above 127. The empty positions should contain the lower- and uppercase versions of ‘eth’ and ‘thorn’, the ‘nj’ ligature, and a non-slanting version of the pound sterling sign; the dc fonts have these positions filled up.

in creating virtual fonts if your drivers can’t handle them.

If you check in *The TEXbook*, Appendix F, you may realize that the roman, italic and typewriter fonts have different layouts; therefore, when you create virtual fonts you must give your programs this kind of information. Just to give an example let’s create the virtual font `emr10` starting from the standard real font `cmr10`:

1. Run `tftovp` by issuing the following command

   ```
   tftovp -rm cmr10.tfm emr10.vpl
   ```

   after having set things up so that the necessary `.tfm` files are in the default directory; the best thing to do is to issue the command in the directory where you have all the `.tfm` files.

   This action creates a virtual property list file, with extension `.vpl`, that contains all the information on the size of every character, the ligatures, the glyphs that are made by superposition of glyphs taken from several other real fonts. Of course, for other cases you might be obliged to use the command `tftovp` in its full glory with all the other options fully spelled out, but this simple example is sufficient for giving the idea of the whole procedure.[1]

2. Now run `vptotf` in this way

   ```
   vptotf emr10
   ```

   obtaining the `.tfm` file (LA)TEX needs for its font selection and the virtual font file (extension `.vf`) that the driver needs for using the virtual font; move this latter file into the directory where the driver(s) expect to find virtual files.

3. The `.vpl` file is no longer needed, so it can be deleted.

Of course you might automate this simple procedure by writing a command (or script) file that performs all these operations with a minimum of human intervention.

If you have `.afm` files for PostScript fonts, you can do similar operations in order to use such outline fonts, but maybe leave that for when you have more experience.

2.2 Getting along with ordinary cm fonts

If you do not have the programs mentioned in the previous subsection and/or your drivers do not handle virtual fonts or do not handle fonts with more than 128 characters, or if you have decided that you do not want to use virtual fonts (for example for portability reasons), do not give up! It is still possible to redefine the accent macros so as to make them a little smarter, i.e. so that they introduce automatically an `\hz` command before and/or after the letter they operate upon, depending on the nature of the following character.

You should be able to perform an anonymous `ftp` to my site:

```
ftp ftp.polito.it
Username:  anonymous
Password:  your e-mail address
cd /pub/tex/polito/hyphens
```

where you can fetch the three files `polyglot.tex`, `polyglot.sty` and `polyglot.doc`. You can use the

[1] Apparently nobody noticed or took care that none of the options available for the `tftovp` command handles the caps-and-small-caps fonts; for such fonts, some editing of the ASCII virtual property list file is necessary before going to the next step, but you’ll produce virtual caps-and-small-caps fonts when you have gained a little experience with virtual fonts. It’s a good idea to use your editor to explore the property list files: you get to understand a lot of things about TEX that are not written in any book, or are presented in such a way that the reader can’t really understand them.

second one as a LaTeX option, so that you can process the third one as a LaTeX document and get all the information about the use and/or initialization of your (La)TeX programs with the `polyglot` macros.

In the same directory you will find the "poor man" hyphenation patterns for several languages; these are labeled "poor man" because they do not consider accented letters and leave the task of hyphenating to the "intelligent" accent macros. Such macros do their best, but of course they cannot perform as well as a (La)TeX system correctly set up with dc or em fonts for handling multiple languages. Nevertheless, I did use such a "poor man" implementation for a while, and I have typeset several documents in different languages with almost no human intervention while getting error-free justified text with hyphenated words, in particular in French and in Catalan.[2]

3 Using em or dc fonts

3.1 Extended fonts and TeX

If you use TeX, and you want to use em or dc fonts, you should do the following:

1. copy the file `plain.tex` to another file, and call it `emplain.tex`;
2. edit `emplain.tex` so that it also preloads the roman, italic, etc., em or dc fonts corresponding to the roman, italic, etc., fonts already loaded; for example, add the lines:

   ```
   \font\etenrm = dcr10
   \font\etensl = dcsl10
   ...
   ```

   or

   ```
   \font\etenrm = emr10
   \font\etensl = emsl10
   ...
   ```

3. modify the definitions for `\rm`, `\sl`, `\it`

   ```
   \def\rm{\fam\z@\etenrm}
   \def\sl{\fam\slfam\etensl}
   \def\it{\fam\itfam\etenit}
   ...
   ```

3.2 Extended fonts and LaTeX 2ε

LaTeX 2ε is already pre-set for use with dc fonts, although the default encoding scheme is the "old" 128-glyph one. Therefore, before setting any text with dc fonts it is necessary to declare the extended `T1` encoding by means of the declaration

```
\renewcommand{\encodingdefault}{T1}
```

in the preamble of your document.

But the above operation is suitable only when you run LaTeX 2ε as a regular program that has already been initialized. Its ability to handle hyphenation patterns in one or more languages derives from its initialization which must be executed according to the procedure described for your particular implementation of TeX; in the following sections, an example is given, but your particular implementation might require some variations.

Regardless of the implementation, though, you need to set up or modify a file named `hyphen.cfg` which is intended for loading several hyphenation pattern files for the corresponding languages. As far as my experience is concerned, the LaTeX 2ε base package obtainable from CTAN states that this task is reserved for TeXperts and a special documentation file is offered to such experts for the task. Unfortunately this file does not say much and several actions must be invented by the user. Two points are to be followed with special care:

1. The command `\newlanguage` is defined to be an `\outer` one so that it cannot be used within the main argument of the command

   ```
   \InputIfFileExists{<A>}{<B>}{<C>}
   ```

   where `<A>` is the file name containing the patterns for a particular language, `<B>` the actions to be taken before inputting the pattern file, and `<C>` the actions to be taken if the pattern file does not exist or can't be found along the default or the specified paths. Therefore, in order to load French patterns, for example, it is necessary to spell out the loading commands:

   ```
   \newlanguage\l@french
   \InputIfFileExists{frhyph}{%
   \language\l@french
   \lccode`\'=`\'
   % etcetera
   }{%
   \errhelp{The configuration for
       hyphenation is incorrectly
       installed.^^J%
       If you don't understand
       this error message you need
       to seek^^Jexpert advice.}%
   \errmessage{OOPS! I can't find
       any hyphenation patterns for
       French.^^J \space Think of
       getting some, or the latex2e
   ```

[2] I mention these two languages (of the eight that my box can handle) because they were used to typeset formal and official documents undersigned by the Rectors/Directors of the Polytechnics with which we made agreements.

```
    setup will never succeed.}%
    \@@end
}
```

(The actions to be taken for a nonexistent or unreachable French pattern file are copied [and slightly edited] from the sample `hyphen.cfg` file that is included in the base package.)

2. Before inputting any pattern file that contains special characters, it is necessary to map the accent macros and their arguments to the charcodes of the corresponding special characters and to define their lowercase codes.

   In fact, the ability of LaTeX 2ε to deal with extended codes through sophisticated accent macros is applicable only during regular typesetting runs, not during the manipulation and digestion of hyphenation patterns. For the latter, simple and direct macros must be designed such as the following:

   (a) For the special characters for which standard commands are available, such as `\o` for ø and `\ss` for ß, it is sufficient to prepare declarations of the following form:

```
\def\ss{^^ff}\lccode"FF="FF
\def\SS{^^df}\lccode"DF="FF
\def\ae{^^e6}\lccode"E6="E6
\def\AE{^^c6}\lccode"C6="E6
\def\oe{^^f7}\lccode"F7="F7
\def\OE{^^d7}\lccode"D7="F7
\def\o{^^f8} \lccode"F8="F8
\def\O{^^d8} \lccode"D8="F8
\def\i{^^19} \lccode"19="19
\def\j{^^1a} \lccode"1A="1A
\def\aa{^^e5}\lccode"E5="E5
\def\AA{^^c5}\lccode"C5="E5
\def\l{^^aa} \lccode"AA="AA
\def\L{^^8a} \lccode"8A="AA
```

   (b) For the other characters — those that carry a diacritical mark — it is better to resort to intermediate macros, some of which map the accent macro and character to a single control word, and some for defining the meaning of such control words. The whole trick is accomplished as follows: first you define the control word mappings

```
\def\'#1{\csname @ac@#1\endcsname}
\def\`#1{\csname @gr@#1\endcsname}
...
```

   and so on, with the prefixes that are listed in Table 2 for the other diacritical marks. Then, for each of the approximately one hundred diacriticized characters, you must set up declarations such as these:

```
\catcode`\^^a0=11 % letter  \u{a}
\lccode"A0="A0    % lc code
\def\@u@a{^^a0}   % ctrl word
%
\catcode`\^^80=11 % letter  \u{A}
\lccode"80="A0    % lc code \u{a}
\def\@u@A{^^80}   % ctrl word
%
...
```

   and so on for the remaining special characters whose codes can be deduced from Table 1.

   (c) Most important, steps 1 and 2 must be confined within a group together with the pattern file input command, so as to keep such declarations local to the sole group where patterns are being manipulated and digested.

3. After closing the group mentioned in the above item, it is wise to declare the default language, the default encoding (so that you do not need to declare it in every preamble of every document), and the relevant parameters for the leftmost and rightmost word fragments:

```
\language=0
\lefhyphenmin=2
\righthyphenmin=3
\def\encodingdefault{T1}
```

   This done, you are ready to run the initializer.

3.3 Extended fonts and LaTeX2.09

If you are still using LaTeX2.09 or you want to have a LaTeX2.09-compatible version, do the following:

1. copy the file `lplain.tex` to a new file, say, `emlplain.tex`;
2. edit `emlplain.tex` by replacing the line `\input lfonts` with `\input emlfonts`
3. copy `lfonts.tex` into `emlfonts.tex`;
4. edit `emlfonts.tex` by adding a line that loads a dc or an em font for every text font already loaded; as shown here (original lines are marked with `<--`):

```
\font\fivrm =cmr5  % roman   <--
\font\efivrm=dcr5  % ex. roman
...
\font\elvrm =cmr10\@halfmag % roman <--
\font\eelvrm=dcr10\@halfmag % ex. roman
...
\font\frtnrm =cmr10\@magscale2 % rom <--
\font\efrtnrm=dcr10\@magscale2 % ex. rom
...
```

5. modify the definitions of the font changing commands for all point sizes; e.g. for the ten-point size, search for the definition `\def\xpt` and change:

   `\def\prm{\fam\z@\tenrm}%`

   to

   `\def\prm{\fam\z@\etenrm}%`

   doing the same for all the other font selections.

When you edit `emlfonts.tex`, near the end of the file, you should comment out the definition of `\$`, because with dc or em fonts the dollar sign always has the correct shape.

The procedure seems very complicated but really it amounts to just some time spent in careful repetitive editing: duplicating lines, replacing `cm` with `dc` and adding an `e` in the proper places.

Another point must be kept in mind: the program `tex` has a finite memory for holding font information. If you have made all the modifications explained above, you end up with a LaTeX format where 106 fonts have been preloaded. This might be too much for your "small TeX", but presents no problems to the "big TeX" implementations.

You also need to edit your newly created files `emplain.tex` and/or `emlplain.tex` and replace

```
\input hyphen
```

with

```
\input lhyphen
```

unless your `lplain.tex` file is dated 31 March 1992 or later, in which case it may already have been done.

4 Hyphenation patterns

The best way to have the proper hyphenation patterns for the languages one chooses to use is to retrieve them from the CTAN archives. If you have access to the Internet, just `ftp` to the nearest CTAN archive and explore the directory `/tex-archive/languages`. Select the directory corresponding to the chosen language, and hopefully the proper set of hyphenation patterns is already there.

If you do not have access to the Internet, you probably know somebody who does. If you don't, ask the TUG office for a diskette containing what you can't otherwise obtain, and pay what the TUG office will charge you. If the whole procedure seems too complicated, just consult the advertising pages of any issue of *TUGboat* and choose the vendor who can provide you with what you desire; the vendor prices are higher but generally you get a full set of diskettes with an `install` program that saves you all the burden of retrieving, initializing, etc.

But what happens if you decide to use a language for which no hyphenation patterns have been prepared? This is most unlikely. If you read the paper by Sojka and Ševeček [6], you will find a table where 38 hyphenation pattern files are examined; they deal with 32 different languages that include both flavors of English (US and UK), modern and classical Latin and Greek, Esperanto, and most modern European and North and South American languages. Besides Greek, patterns exist also for languages that do not use the Latin alphabet (with or without diacritics), such as Russian, and possibly for less known regional languages.

But what if you are unlucky — the patterns you are looking for do not exist, or they exist but are unreachable, or they are too large for the capabilities of your `tex` program ... In this case you yourself must create a file containing your patterns; you must carefully read Appendix H of *The TeXbook* and, of course, you must know the "strange" language you want to use, or at least have a perfect knowledge of its grammar and hyphenation rules. It's not too difficult, with or without the use of the program `patgen`, but this topic is sort of self-contained, so I leave it for another tutorial [1].

So, from now on, we assume that you have all the pattern files you need. You should at this point edit (or create) the file `lhyphen.tex` to read:

```
% File lhyphen.tex created on ...
% by ...
% It loads the hyphenation patterns
% for the following languages:
% 0) US english
% 1) italian
% 2) french
% 3) ...
%
\input chardefs
%
\language=0 \chardef\l@english 0
\input hyphen
%
\newlanguage\l@italian
\language\l@italian
\input ithyph
%
\newlanguage\l@french
\language\l@french
\input frhyph
%
% and so on
%
% Default values
\language\l@english
```

```
\lccode‘’= 0
\lefthyphenmin=2
\righthyphenmin=3
%
\endinput
```

If you are following the "poor man" procedure, the `chardefs.tex` file to be input might read simply like this:

```
\lccode‘’=‘’
\catcode’33=11 %\oe
\lccode’33=’33 %\oe
\let\oe=^^1b
...
```

and similar definitions and code assignments for all the other special ligatures or diacriticized characters present in the cm fonts that are necessary in the languages you are going to use.

The apostrophe should receive an `\lccode` different from 0 because it should not interrupt TEX words of the type *quest'anello*, *l'approbation*, *s'organizzen* in languages such as Italian, French, or Catalan (where vocalic elision is marked with an apostrophe) — remember the definition of a TEX word: "... characters..., with a non-zero `\lccode`,...". This is one of the most frequent requests for help I receive from Italian users, who forget to `\lccode` the apostrophe and complain about (LA)TEX not hyphenating after such a sign.

If you are going to use extended fonts (dc or em) you need a `chardefs.tex` file that defines a set of macros for changing such sequences as `\u{a}` (ă) into the numerical code of the diacriticized letter (in this case `’240` or `"A0` or `160`), while assigning a non-zero `\lccode` to the character in question. Such a file, in other words, should contain things similar to those that have been described for LATEX 2_ε.

Such macros are simple but numerous when you want to have a complete map to all 102 diacriticized letters of the Cork encoding and to the 13 special characters ß, œ, æ, ø, å, ł, ı, and ȷ, with their uppercase possible counterparts (plus the apostrophe):

```
% Save current @ catcode and ...
\chardef\atcatcode=\the\catcode‘\@
% ... make it a letter.
\catcode‘\@=11
\def\ss{^^ff}\lccode"FF="FF\uccode"FF="DF
\def\SS{^^df}\lccode"DF="FF\uccode"DF="DF
\def\ae{^^e6}\lccode"E6="E6\uccode"E6="C6
\def\AE{^^c6}\lccode"C6="E6\uccode"C6="C6
\def\oe{^^f7}\lccode"F7="F7\uccode"F7="D7
\def\OE{^^d7}\lccode"D7="F7\uccode"D7="D7
\def\o{^^f8} \lccode"F8="F8\uccode"F8="D8
\def\O{^^d8} \lccode"D8="F8\uccode"D8="D8
\def\i{^^19} \lccode"19="19\uccode"19="49
\def\j{^^1a} \lccode"1A="1A\uccode"1A="4A
\def\aa{^^e5}\lccode"E5="E5\uccode"E5="C5
\def\AA{^^c5}\lccode"C5="E5\uccode"C5="C5
\def\l{^^aa} \lccode"AA="AA\uccode"AA="8A
\def\L{^^8a} \lccode"8A="AA\uccode"8A="8A
\lccode‘’=‘’
```

The hexadecimal codes appearing in the above definitions can be found in Table 1, where the octal and hexadecimal codes for all the other diacriticized characters can also be found. The uppercase codes are specified so that the case of such special letters can be properly changed.[3]

Besides the above macros `chardefs.tex` must also contain the character mappings necessary for the cases when accent macros are used; in the CTAN archives you can find a file, `compatible.tex`, that contains intermediate macros for these mappings. They look like this:

```
\def\’#1{{\expandafter
  \ifx\csname @ac@#1\endcsname\relax
    {\accent19 #1}%
  \else
    \csname @ac@#1\endcsname
  \fi}}
\def\‘#1{{\expandafter
  \ifx\csname @gr@#1\endcsname\relax
    {\accent18 #1}%
  \else
    \csname @gr@#1\endcsname
  \fi}}
...
\def\~#1{{\expandafter
  \ifx\csname @til@#1\endcsname\relax
    {\accent’176 #1}%
  \else
    \csname @til@#1\endcsname
  \fi}}
\def\"#1{{\expandafter
  \ifx\csname @um@#1\endcsname\relax
    {\accent’177 #1}%
  \else
    \csname @um@#1\endcsname
  \fi}}
```

The trick is this: any accent macro (for example, the one for the acute accent), operating, say,

[3] This is not necessary with LATEX 2_ε because definitions are local to the section where patterns are handled. On the other hand, remember to avoid groups while specifying these codes with LATEX2.09, otherwise you lose the possibility of treating words with special characters in the proper way.

on the letter 'a', maps to the protected[4] internal control word `\@ac@a`, if this word is defined; otherwise, it operates as a regular accent macro with cm fonts. If you are using the extended fonts and the set-up I am describing, you might as well simplify such macros to:

```
\def\'#1{\csname @ac@#1\endcsname}
\def\`#1{\csname @gr@#1\endcsname}
...
\def\~#1{\csname @til@#1\endcsname}
\def\"#1{\csname @um@#1\endcsname}
```

The whole set of prefixes of such control words is summarized in Table 2. Actually, there are no LaTeX macros for the ring accent and the ogonek diacritical mark;[5] if you need to use them, you can define `\r` and `\g` to map to the proper control words by means of the prefixes shown in Table 2.

Next you must assign every such control word to an extended character and then assign that character the proper `\lccode` and `\uccode`, an operation that is lengthy because of the number of characters, but at least is repetitive; all such assignments are of this type:

```
\catcode`\^^a0=11 % letter  \u{a}
\lccode"A0="A0    % lc code
\uccode"A0="80    % uc code \u{A}
\def\@u@a{^^a0}
%
\catcode`\^^80=11 % letter  \u{A}
\lccode"80="A0    % lc code \u{a}
\uccode"80="80    % uc code
\def\@u@A{^^80}
%
...
```

The CTAN archives contain a file called `extdef.tex` that has definitions similar to the ones above, but resorts to two macros:

`\csubinverse` and `\charsubdef`

that map em/dc-font extended characters to corresponding cm-font accent macros + characters, and vice versa. These macros should not be necessary if you consistently use only extended fonts.

On the other hand, if you have a keyboard with national characters (and you pay some attention by editing your files before you send them to colleagues who might not have the same keyboard) you could reduce your keying if you map the input codes directly to the corresponding extended TeX codes; for this you might add to your `chardefs.tex` file some definitions of the form:

```
\catcode`\à=13 \def à{\@gr@a}
\catcode`\á=13 \def á{\@ac@a}
...
\catcode`\ñ=13 \def ñ{\@til@n}
```

Remember to end the file by restoring the right `@` catcode:

```
\catcode`\@=\atcatcode
```

If you spend the time necessary to create this `chardefs.tex` file from scratch, to explore the CTAN archives, and to put together the various parts, you understand why vendors charge you a reasonable price for selling TeXware that everybody could otherwise get at no cost!

grave	`@gr@`	caron	`@v@`
acute	`@ac@`	breve	`@u@`
circumflex	`@hat@`	macron	`@eq@`
tilde	`@til@`	dotaccent	`@dot@`
dieresis	`@um@`	cedilla	`@c@`
hungarumlaut	`@H@`	ogonek	`@og@`
ring	`@r@`		

Table 2: Control-word prefixes for accent-macro mapping

5 Initialization

Now you have all the necessary pieces of information to create the formats `latex.fmt` with LaTeX 2_ε, `emplain.fmt` with extended font plain TeX, and `emlplain.fmt` for extended font LaTeX2.09.

It's time to run `initex`, the TeX initialization program, the only one that can chew and digest the hyphenation patterns and store them into memory in a special way so that during a regular TeX run the hyphenation process proceeds efficiently.

Depending on your software implementation, this initialization program may be a different `.exe` (i.e. there are both `tex.exe` and `initex.exe`), or the initialization may be an option to the regular procedure, or the executable module may behave differently depending on the name used to invoke it. Again, depending on the operating system, you might also include the arguments to the command within double quotes. So this simple initialization task may prove to be more difficult than it should.

On my DOS personal computer, with the various implementations I have used, I had to follow three different approaches; however, command files come in handy for doing the whole job without bothering too much about the details. You have to:

[4] 'Protected' in the sense that it contains the character `@`, which is not a letter during normal (LA)TeX operation, so that it is impossible to inadvertently redefine it.

[5] Actually the ring accent is used only on the letters 'a' and 'u', so that the standard LaTeX macros `\aa` and `\AA` are sufficient for the former, but you still need something for using the latter.

1. set up the environment variables according to the documentation for your particular implementation of TEX;
2. set up the search paths, if necessary;
3. invoke the initialization program with the proper syntax and with the proper arguments;
4. `\dump` the format;
5. move the format to the directory where TEX expects to find format files;
6. possibly preload the format so as to create a specific executable program.

Steps 1 and 2 are very much system- and software-dependent, so please check your documentation very carefully. Below I show how I would do the job for LATEX2.09, by means of a C-source derived `tex` program named `ctex` when it operates as a regular `tex` program, and `cinitex` when it operates as an initializer program.[6]

1. I set up the following environment variables (but remember that the names of my directories do not necessarily match yours):

   ```
   SET TEXFORMATS=C:\TEX\ctexfmts
   SET TEXPOOL=C:\TEX\ctexfmts
   SET TEXFONTS=C:\TEX\fonts
   SET TEXINPUTS=.;C:\TEX\inputs
   ```

   With other implementations it might be possible to set environment variables that control the amount of memory that the program assigns to the various operating parts.
2. I decide if the executable `cinitex.exe` is already in the proper directory; if not, I rename the executable `ctex.exe` to that name

```
if exist C:\TEX\cinitex.exe goto runinitex
rename C:\TEX\ctex.exe cinitex.exe
```

3. Then I produce the format and `\dump` it in one step:

   ```
   :runinitex
   C:\TEX\cinitex %1 \dump
   ```

   In the above command `%1` is the name of the format I want to produce; in our examples it might be `emplain` or `emlplain`.
4. Next I move the format file to the proper directory

   ```
   move %1.fmt C:\TEX\ctexfmts
   ```
5. Finally I reset the program name to `ctex.exe`

   ```
   rename C:\TEX\cinitex.exe ctex.exe
   ```
6. At this point, with some implementations of `tex`, it might be possible to run a preloading facility that creates an executable image of the program with the format preloaded; it speeds up the preliminary operations of a regular TEX run a little, but it is not a real necessity; many implementations do not have this facility.

That's all. For running (LA)TEX, another command file sets up the same environment variables (in case they were not set or had been reset), and invokes the executable with:[7]

```
ctex &emlplain %1
```

where, as usual, `%1` is replaced by the `.tex` file name you want to process.

During initialization you might experience a number of error messages; try to understand which is the cause and exit the initializer with `x` at the program prompt. Some possible causes include:

1. `initex` did not find some `.tex` file(s): check that all the necessary files are on a TEX search path conforming to the `TEXINPUTS` environment variable.
2. `initex` did not find some font files: check for spelling errors in the parts you have edited; check that all `.tfm` files are in the directory(ies) identified by the `TEXFONTS` environment variable.
3. `initex` complains about the `tex.pool` file:[8] the file is missing or the file in the directory pointed at by `TEXPOOL` does not belong to the particular TEX implementation you are using; retrieve the correct `tex.pool` file and be sure to put it in the correct directory.
4. `initex` complains about some undefined control sequence: check the name of the control sequence and correct your spelling in the files you have edited.
5. `initex` complains about duplicate patterns: press `<Enter>` and keep going, but if the problem shows up again, you'd better get out with `x`. In any case, check your hyphenation files and your `lhyphen.tex` file; you may have forgotten to issue the command `\newlanguage` or most probably you made a spelling mistake. If you made your own patterns the possibility of

[6] Other implementations may have `initex` as the initializer, `virtex` as a "virgin" version of TEX (that is, a program without any format preloaded), `tex` as a version with `plain.fmt` preloaded, and `latex` as a version with `lplain.fmt` preloaded. In other systems, particularly UNIX, all these names are aliases that call the same executable which behaves differently according to the name by which it has been called.

[7] Change `emlplain` to `emplain` if you want to use plain TEX instead of LATEX.

[8] On DOS platforms the name is `tex.poo`.

having inadvertently duplicated some patterns is normal, but must be corrected.

6. `initex` complains about memory limitations: if you are using a "big TEX" implementation based on a C-source, this should not happen. If it does, either you want to go into the Guinness book of records for the largest number of languages treated at the same time, or you have bad patterns, or your C-source program did not work properly. If your program comes from a Pascal source and you have the source code, after having properly checked that you are not going into the Guinness book, and that your patterns are OK, you could carefully modify the memory declarations in the source code and recompile and link the program. I hope you know what you are doing. If you do not have the source code, but you have an implementation (like `sb39tex`) that allows you to exercise some control over the memory usage, check the documentation and proceed accordingly.

With respect to memory limitations, specifically hyphenation memory limitation, TEX has two memory areas, the *trie* and the *ops* ones, the former being dedicated to holding the patterns (all the patterns of all the languages that have been loaded) and the latter to holding the "branch" information of the pattern structured lists. The greater the number of patterns, the greater the trie memory occupied; the more complex the pattern structure, the greater the ops memory occupied.

When you run `initex`, the `.log` (or `.lis`) file documents all the relevant information about the run; in particular, towards the end you will find a listing that looks like this:

```
14 hyphenation exceptions
Hyphenation trie of length 8172
         has 407 ops out of 750
  23 for language 6
  15 for language 5
  15 for language 4
  25 for language 3
  110 for language 2
  38 for language 1
  181 for language 0
```

from which you can obtain important information about the memory occupation dedicated to multilanguage issues:

1. the 14 hyphenation exceptions are those that come with the US-English `hyphen.tex` file, originated by Liang and Knuth and described in *The TEXbook*, Appendix H. The other languages I loaded do not have hyphenation exceptions because this is my policy when I prepare my patterns.

   There is nothing wrong with hyphenation exceptions; simply they are better suited for a non-flexive or moderately flexive language, such as English, than for flexive languages such as all the Romance ones. In English, for a noun you have two forms, for an adjective one form, for a verb four or five forms. In Italian nouns require two forms, adjectives up to four, verbs approximately 60 and this does not include all the possible agglutinations of enclitic pronouns. For French, Spanish, Portuguese, Catalan, Romanian, and Latin the situation is similar, so that if a hyphenation exception involves the stem of a verb, you may have to input some 60 entries (at least) in the `\hyphenation` argument!
2. US-English hyphenation patterns occupy 6075 trie memory words and 181 ops; the standard size of the trie memory (in a regular-sized TEX implementation) is 8000 words; French comes second in complexity, requiring from 1122 to 1433 trie memory words[9] and 110 ops; the other Romance languages require on the average 600 trie memory words and 25 ops. The ops numbers are simply additive while the trie memory words are not, because the cited numbers include some undocumented overhead, so that the final trie memory occupied is smaller than the sum of the single language trie memory occupied. But during the `initex` run you need the availability of at least the sum of the single language occupations, so that the program can massage the patterns and compress them in the proper way.

   When you have to control the memory occupation, you must keep these numbers in mind in order to understand possible complaints by `initex`.
3. The ops numbers give you a fairly good idea of the complexity of the hyphenation rules for a given language as they are translated into TEX code. In my experience, German patterns appear to be the most complicated, requiring 9980 trie memory words and 281 ops; this is the set of patterns indicated as DE_{max} in Sojka and Ševeček [6],[10] and if you have to use German

[9] Depending on the presence of extended characters; the numbers refer to my poor-man or extended-character patterns, respectively.

[10] However, they report 255 ops while my `cinitex` executed 281; perhaps the files we examined are not exactly the same.

you'd better have a "big TEX" implementation with a large trie memory size.

6 Language-dependent macros

The best thing to do is to retrieve the babel package; this is the finest set of multilanguage macros I know of and I recommend it to everybody. My polyglot macros are also an acceptable "poor man" alternative for those who have stuck to cm fonts and relied on intelligent accent macros. The macros should be divided into three categories:

1. language selection;
2. label selection;
3. style selection.

If you want to make your own macros, because you are going to use your multilanguage implementation in a restricted or in a special way, you can do the following.

6.1 Language selection

In the lhyphen.tex file all languages were identified by a literal name mapped to an internal number of the form \l@*language*; this allows you to set up simple macros such as these:[11]

```
\def\Lang#1{\expandafter
   \language\csname l@#1\endcsname
   \csname #1settings\endcsname}
%
\def\englishsetting{\lccode‘’=0
\righthyphenmin=3}
%
\def\italiansettings{\lccode‘’=‘’
\righthyphenmin=2}
%
\def\frenchsettings{\lccode‘’=‘’
\righthyphenmin=3}
%
% and so on
%
```

You change hyphenation rules simply by issuing commands of the type

```
{\Lang{french} ... % french text
}                  % end french
```

without forgetting the group braces so that the action is confined to the enclosed group alone.

In LATEX you can use the above definitions also as environments:

```
\begin{Lang}{french}
...                % french text
\end{Lang}
```

or you might prefer to define a new environment:

```
\newenvironment{French}{\Lang{french}}{}
```

so that you can type:

```
\begin{French}
...                % french text
\end{French}
```

and thereby following standard LATEX markup style.

6.2 Label selection

What has been described in the previous subsection is suitable for setting short passages of one language within a text composed in another language; for example, French extracts (properly hyphenated in French) in an English-language essay (with citations in an appropriate English style).

If you want to write a whole document in another language you should be able to change "Chapter" into "Chapitre", "Table" into "Tableau", and so on, with a single command. Fortunately, as of 31 March 1992, these words are no longer hardwired into the LATEX styles; they are contained in control sequences with self-explanatory names. So, together with the language settings, you need a \setcaptions macro of the following type:

```
\def\setcaptions#1{%
\csname #1captions\endcsname}
%
\def\frenchcaptions{%
\def\refname{R\’ef\’erences}
\def\abstractname{R\’esum\’e}
\def\bibname{Bibliographie}
\def\chaptername{Chapitre}
\def\appendixname{Annexe}
\def\contentsname{Table des mati\‘eres}
\def\listfigurename{Liste des figures}
\def\listtablename{Liste des tableaux}
\def\indexname{Index}
\def\figurename{Figure}
\def\tablename{Tableau}
\def\partname{Partie}
\def\enclname{P.˜J.}
\def\ccname{Copie \‘a}
\def\headtoname{A}
\def\pagename{Page}
%
\def\today{\ifnum\day=1\relax
1\/$ˆ{\rm er}$\else\number\day\fi
\space\ifcase\month\or
janvier\or f\’evrier\or mars\or
avril\or mai\or juin\or juillet\or
ao\ˆut\or septembre\or octobre\or
```

[11] \lefthyphenmin is supposed to maintain the default value of 2; of course some settings might change this value, but if you choose to do so, you should insert an appropriate setting for every language.

```
novembre\or d\'ecembre\fi
\space\number\year}
}
%
% and so on for the other languages
%
```

With a LaTeX2.09 document you can start this way:

```
\documentstyle{book}
\Lang{french}\setcaptions{french}
\begin{document}
...
\end{document}
```

The language selection macros and the caption definition commands could also be incorporated into `hyphen.cfg` for LaTeX 2_ε, and into `lhyphen.tex` for LaTeX2.09, so that they remain available with any document type and you need not specify a particular option or package every time you start a new document.

6.3 Style selection

If you want your composition style to be correct, all the way down to respecting the typographic traditions of another country, then you should rely completely on the babel macros.

The most apparent differences consist in these points:

1. In France (and perhaps in some other countries) French spacing is normally used; French spacing leaves the same amount of white space after all punctuation marks, but leaves some thin space before the "tall" punctuation marks such as the colon and semicolon, the question and exclamation marks, the parentheses,[12] the quotation marks; such spaces cannot split at the end of the line. All these punctuation marks must be made active in text mode so that they introduce the right amount of white space, or, conversely gobble extra white space.
2. The quotation marks are the most variable in different countries; in the US high quotation marks with the shape of normal or reversed commas are used; in France and many other countries guillemets are used; in Italy we use both types. In some countries open quotation marks are identical to closed quotation marks but are lowered; in other countries they are reversed, that is left quotation marks are used for closing a quotation instead of opening it. And so it goes.

   All these marks are present in the extended fonts, although in the em fonts the guillemets are far from optimal; dc fonts and PostScript fonts have perfect shapes. But in any case suitable macros must be set up correctly in order to insert the proper quotation marks.
3. Ordinal numbers are another point of difference; in some countries a digit is immediately followed by the desinence or the final letter of the corresponding spelled-out ordinal; in other countries this literal ending is separated by a period; in yet other countries this ending is set as an exponent (sometimes underlined). Flexive languages may have different endings for masculine and feminine, singular and plural.

   Elsewhere, ordinals are expressed by roman numerals; the lowercase roman letters are customary in English, and this habit has gained wide acceptance in many places, while good style in many countries requires the use of uppercase roman letters only, possibly from a small caps font.

For these sorts of style problems, don't do it the do-it-yourself way; it's too difficult (unless you are a book connoisseur, of course). It's better to rely on the babel package, which, although it was created and is maintained by one man, J. Braams, benefits from continuing suggestions and constructive criticism by all members of the TeX community, with the result that the package is constantly being upgraded and always becoming better.

7 Conclusion

This long tutorial on multilanguage typesetting has tried to focus on some of the problems concerning setting texts in different languages by means of a single program, TeX or LaTeX. Configuring the program to handle several languages is not that difficult, but it does require patience, good knowledge of the software, a long week-end... But when you've set up the program the proper way, you will enjoy it much more than before.

References

[1] Beccari C., Oprea R., Tulei E., "How to make a foreign language pattern file: Romanian", *TUGboat*, 16(1):31–42, March 1995.

[2] Braams J., "babel, a multilingual style-option system for use with LaTeX standard document styles", *TUGboat*, 12(2):291–301, June 1991. Update: *TUGboat* 14(1):60–62, April 1993.

[3] Goossens M., Mittelbach F., Samarin A., *The LaTeX Companion*, Addison-Wesley, Reading, Mass., 1994.

[4] Knuth D.E., *The TeXbook*, Addison-Wesley, Reading, Mass., 1990.

[12] The thin white space goes *after* the open parenthesis.

[5] Lamport L., *LaTeX A Document Preparation System*, Addison-Wesley, Reading, Mass., 1986. Second edition, dealing with LaTeX 2$_\varepsilon$, 1994.

[6] Sojka P., Ševeček P., "Hyphenation in TeX — Quo Vadis?", *Proceedings of the Eighth European TeX Conference* (Sept. 26–30, 1994, Gdańsk, Poland), 59–68.

◇ Claudio Beccari
Dipartimento di Elettronica
Politecnico di Torino
Turin, Italy
Email: beccari@polito.it

How to make a foreign language pattern file: Romanian

Claudio Beccari, Radu Oprea and Elena Tulei

Abstract

In this tutorial we show how to make hyphenation patterns for a language when you know the grammatical rules for hyphenation (if they exist). We also discuss some points related to typographical hyphenation when compared with grammatical syllabification.

1 Introduction

In this tutorial we complete the argument of multilanguage typesetting that we started in the previous article [3]; there the problem of pattern generation had been omitted because this topic is sort of self contained and deserves its own discussion.

The problem of creating patterns suitable for TEX's hyphenation algorithm is dealt with in Appendix H of Knuth [10], where a certain statement may induce TEX users to abstain from creating pattern tables "since patterns are supposed to be prepared by experts who are well paid for their expertise."

In a way this is desirable, so that pattern tables for the same language are not created continuously, leading to multiple, incompatible, different, dubious, erroneous pattern lists and frustrated users unable to find a reliable pattern set for the language they want to use.

In another sense, pattern creation is essential for the wide-spread dissemination of TEX, which is now being used at least for all the languages examined by Sojka and Ševeček [12], and certainly also for many other modern and ancient languages. An indirect proof is given by the existence of fonts for exotic languages, fonts that could not be of any use if nobody set texts in those languages — setting text implies breaking words at the end of the line, so that some sort of hyphenation patterns must be used.

When the first author of this article started his interest in hyphenation, he had to make up for patching or creating new patterns for his site because the patterns available for Italian at that time did not comply with national regulations and, to be honest, were quite poor. He ended up with new patterns that were published in *TUGboat* [2] and then made their way into the CTAN archives. Recently, version 4.1, which greatly improves hyphenation of technical terms, was submitted to the archive managers.

Due to the presence of several foreign visiting professors and undergraduate and graduate students in his Polytechnic, Beccari was asked to prepare versions of (LA)TEX suitable for French, Spanish, Catalan, and Romanian; for his own pleasure he added Portuguese. He decided to prepare a single multilingual version of the formats so that users would not have to keep different commands in mind and would have the possibility of switching back and forth between languages. The limitations included the following:

1. programs had to run on a cluster of VMS mainframes running a Pascal-derived "big TEX";
2. a large number of isolated and clustered UNIX workstations running C-derived versions of "big TEX", a countless number of DOS personal computers, which has the most varied hardware and software configurations, equipped with the most unpredictable versions of TEX (from version 1.5 (!) to 3.1415), as well as screen and printer drivers;
3. the DOS personal computers were mostly owned by students, whose limited budgets were not sufficient to upgrade and standardize their equipment.

Beccari therefore had to adopt a "poor man" policy so that `.tex` files edited and tested at home could also be run on the mainframes or the UNIX stations. This implied sticking to regular cm fonts and creating a set of "intelligent" accent macros[1] that could look ahead and do some intelligent discretionary break insertion. The results were satisfactory for all the cited languages except Romanian, where TEX runs yielded poor results (many overfull

[1] By "accent macro" we mean every command that places a diacritical mark over or under any letter.

and underfull hboxes) with line widths shorter than ca. 100 mm.

With the collaboration of the second and third authors (Oprea and Tulei), natives speakers of Romanian, we decided to prepare Romanian hyphenation patterns without being confined to cm fonts, yet still referring to the extended dc or em font setup, according to the Cork encoding (see Beccari, elsewhere in this issue [3]).

The attentive reader may ask himself: "Why didn't they simply retrieve the Romanian patterns that Sojka and Ševeček [12] mention had been prepared by Malyshev or by Samarin and Urvantsev?" The answer is simple: the articles mentioned in Sojka and Ševeček deal with Russian; the Romanian patterns are only mentioned in passing; Sojka probably got hold of them, but we did not succeed. Moreover, according to Sojka and Ševeček, these Romanian patterns make up a very large set (4121 patterns) that requires a trie size of 4599 words, which might be too large for a multilingual (LA)TEX implementation where they are supposed to coexist with six or seven other pattern sets (including the US-English patterns that one cannot do without).

2 Grammar vs. typography

Preparing hyphenation patterns for TEX is not a simple grammar exercise; one must know the grammatical rules (if they exist and are not too complicated) of the language for which patterns are being prepared, and one must also know the typographical rules valid for the specific country or for a generic good typographical practice in that language.

It is obvious that typographical hyphenation must comply with grammar rules; it is less obvious that the possible hyphen points *must not* coincide with syllable divisions, but are a subset of the latter. Just to set forth a simple example, in Italian the word *idea* is syllabified in `i-de-a`, but typographically this word has no hyphen points;[2] some of (or all) the syllables may show up again when the word is connected with other words, as in *dell'idea* that is divisible like this: `del-l'i-dea`.

Syllables and typographic divisions differ in another way; in some when reading their mother language, some people feel uncomfortable if they have to read a word split across a hiatus (that, is a pair of vowels that do not form a diphthong); more generally, they feel uncomfortable if the word fragment that goes to the next line begins with a vowel that does not serve as a semi-consonant; this is not certainly the case in English-speaking countries, where a division such as *liq-uid*[3] is not only grammatically correct but also acceptable to the readers — such a division, on the other hand, makes Romance language speakers shiver unpleasantly! With Romance language hyphenation patterns, therefore, vocalic groups should remain undivided, except when semiconsonants are present.

So we have two different concepts, *hyphenation* and *syllabification*: the former deals with typography, while the latter deals with grammar.

A *syllable* is a word fragment that contains at least one vowel or equivalent sound; speakers of the language should be able to pronounce it as an isolated utterance; the set of vowels, diphthongs and triphthongs of a language is specific to that language. Sometimes we are astonished when we see foreign words that apparently do not contain any vowels; this is due to the way we are used to classifying the letters of our alphabet, but there is no doubt that in several Slavic languages the letter *r* plays the rôle of a vowel (*črn* 'black', *smrt* 'death', *Trst* 'Trieste').

The consonants on either side of such sounds may belong to the left or to the right syllable, depending on various rules and the acceptability of pronouncing consonantic clusters by native speakers of a specific language; however, etymological rules may alter this kind of division, which explains why syllabification rules are so different in different languages.

Unluckily TEX has no notion of vowels and consonants, so that one cannot rely on the fact that no (incorrect) breakpoints are executed that isolate groups of consonants; in fact, if you ask TEX to show the hyphen points for *comparands*, for example, while English is the default language, you get `com-para-nds`, which is clearly wrong.

Another point, valid for every language, is the existence of syllabification rules; we dare say that these rules exist for every official language, but what about dialects or regional unofficial languages? It is quite likely that the latter completely lack a standard and recognized set of grammatical rules, and that the language variety might change from village to village both in spelling and word usage. But if you must write a report on philological research dealing

[2] For the moment, let us skip the question of minimum length of the first and last "typographical" word fragments, that TEX deals with the values stored into the internal `\lefthyphenmin` and `\righthyphenmin`.

[3] US hyphenation verified in the Webster dictionary. UK hyphenation, as given by TEX by means of the pattern file `UKhyphen.tex`, is 'li-quid'. Although the UK hyphenation file is huge (55 860 bytes, 8527 patterns, 10 995 trie memory words, 224 ops), it hyphenates the words reported in Appendix H of *The TEXbook* in a funny way.

with such languages, you probably need to set many specimens of text, which means that you need patterns for these languages also.

But even if we examine only official languages, such rules, in particular those regarding syllabification, might be too complicated or might refer to word stress. In the case of complicated rules, the following statement holds true:

too complicated rules = no rules

As for word stress, TEX (or any other program that deals with words instead of numbers) does not know anything about stress, so that it cannot take it into account for hyphenation purposes.

In English (US and UK alike) the rules are quite complicated and refer to stress; the example shown by Knuth in Appendix H of *The TEXbook* regarding the word *record* is typical: the stress is different whether the word is used as a noun or as a verb, therefore the syllabification turns out to be `rec-ord` or `re-cord`! There is no simple way to get around these situations.

Another crucial point is the question of compound words, and prefixes and suffixes. Compound words should preferably be divided at word boundaries, while prefixes and suffixes may be treated in different ways in different languages.

In some languages compound words are very frequent; in other languages, although compound words exist, "compound concepts" are expressed without agglutination — the building of long and complicated compound words from many smaller ones — a feature of Germanic languages. Romance languages prefer constructions that make use of conjunctions and prepositions; English is different yet again, often putting together sequences of words that form a "compound concept" without "gluing" them to one another. For example, in English you have the compound list "manufacturing systems engineering" that in French becomes the prepositional phrase "genie des systèmes de fabrication". In any case, compound words are typical of the jargon found in chemistry, regardless of language, and, no doubt, since chemical names tend to be very long, they should be divided on the word boundaries rather than on syllable boundaries.

Prefixes and suffixes are generally treated in two different ways in different languages: prefixed and suffixed words must be divided according to their etymology, or they may be treated as regular common words, disregarding the presence of prefixes and suffixes. Italian and Portuguese belong to the second class, where prefixes and suffixes may be ignored, while most other languages require etymological division — Romanian belongs to this class.

Another problem with some languages is the fact that hyphenated words change spelling compared to when they are undivided; in German, for example, the orthographic sequence *ck* often gets hyphenated into `k-k`.

All these problems offer the willing pattern creator two alternatives: process a language dictionary with `patgen` or do everything by hand.

`patgen` is a program created by Liang for use with TEX (documentation available from CTAN). You have to feed the program with a large set of hyphenated words (several thousand words) and it will produce the hyphenation patterns that TEX requires. Producing patterns this way is time consuming because you need to create a large file of hyphenated words absolutely without error (either in spelling or hyphenation), but it is the only practicable way when dealing with languages for which the *no rules* statement applies. And this statement must be extended to all languages where stress plays an important rôle in syllabification, and where prefixes and suffixes must be divided *automatically*, according to etymology.

3 Mute vowels

Some languages — French for instance — have mute vowels; that is, vowels that are not pronounced, or are pronounced in an indistinct way. According to grammar rules, these vowels may produce a syllable, but typographic practice avoids splitting words which would leave a mute vowel to start the new line, especially if this happens on the last syllable. For French hyphenation patterns, M. Ferguson [9] explicitly inhibited hyphenation for all mute endings. This is another instance where syllabification and hyphenation are in contrast: if patterns are generated via `patgen`, the syllabified list of words fed to the program must take hyphenation breaks at mute vowels into account.

In any event, we want to stress the point that the word fragments obtained through the process of hyphenation are less than or equal to the number of syllables the word contains. There is no point in measuring hyphenation algorithm performance by counting the number of breakpoints it misses[4] (with a chosen word list), because the point is to measure the number of wrong breakpoints it produces. The

[4] Unless the algorithm is so poor that it misses most breakpoints!

best algorithm should produce no incorrect breakpoints (obviously) but apparently this is not achievable with any algorithm.

Beccari's pattern list for Italian `ithyph.tex`, version 3.5, was supposed to correctly hyphenate all Italian words; after additional checking with dictionaries created by L. Bianchi [6], a version 4.0 had to be produced, in spite of the fact that hundreds of people at his site had used the patterns and no one had ever found an incorrect breakpoint.

4 Compound words

`patgen` can also produce patterns for division of compound words, but the pattern list may become extremely complicated, and since compound technical words are continuously being created, any pattern list becomes obsolete the very moment it is created. We are of the opinion that it is too exacting a request that a computer program provide a complete, fast, accurate, and reliable algorithm suitable for every language and every situation. Some type of manual intervention is therefore necessary; below are some options which have proven useful to us.

(LA)TEX offers the macro `\-` that allows hyphenation in specific points. If you use `\-` within a word, this word can be hyphenated at that and only that position.[5]

Another macro with more flexibility is shown below:

```
\def\hz{\nobreak\hskip0pt \relax}
\def\allowhyphens{\hz\-\hz}
```

This new definition allows you to insert a discretionary break without inhibiting hyphenation in the rest of the word. The only drawback to such a definition is that the name is too long. German TEX users get around this problem by making the double quote character " active. Its definition is quite complicated because it has to do a lot of things in different circumstances with a minimum of keyboarding: the double quote prefixed to a vowel inserts the umlaut (dieresis), while prefixed to certain consonants it inserts the appropriate `\discretionary` command (for example, `"ck` expands to

```
\discretionary{k-}{k}{ck}
```

plus no breaks or zero skips to allow hyphenation in the rest of the word). Among other things, the double quote character can change `"-` into the "soft" discretionary break obtained via `\allowhyphens`, defined above.

Active characters are no problem, provided you add them to the lists of special characters whose "activity' or "specialness" is turned off when you go into verbatim mode. But it is a shame that all the standard input characters obtainable with a US keyboard have already been used for special TEX purposes.

In fact we would have preferred that *a single character* be used in place of the control sequence `\allowhyphens`, so as to speed up keyboarding. Moreover, when using dc fonts, it should be possible to refer to character `"17` (called a *compound word marker*). If such a character could be included within the hyphenation patterns, and if it could be easily inserted into the TEX input file, much keyborading could be saved. Unfortunately the character belongs to the set of the "illegal" ones that (LA)TEX refuses to read.

For a while, Beccari (compare [2]) used the underscore `_` as the single-character command, but other TEXies at his site were so used to using the underscore for other purposes (for example in file names) that his choice was doomed to failure; he had to delete the definition in order to avoid continuous quarreling with his colleagues.

In any case — call it `"-`, `\allowhyphens`, or `_` — this macro is suitable for separating prefixes, not for compound words, whose division should take precedence over syllable division; see the discussion in Sojka and Ševeček [12]. This question of marking the boundaries of compound words is still an open one, and we hope that the LATEX3 team finds a suitable solution.

5 Patterns

According to Appendix H of *The TEXbook*, a pattern is a sequence of lowercase alphabetic letters (and characters of category 12, "other") separated by digits.[6] The meaning of the digits d_i and the letters l_i is as follows: the pattern $d_1 l_1 d_2 l_2 \cdots l_{n-1} d_n$ implies that if the sequence of letters $l_1 l_2 \cdots l_{n-1}$ appears in a word, the hyphenation "weights" between such letters are d_1, d_2, ..., d_n, where odd digits allow hyphenation, even digits inhibit hyphenation and, if two or more (different) patterns specify different weights between the same letters, the highest digit prevails. The digit `0`, being the smallest, is optional and may generally be omitted.

A pattern list is a sequence of different patterns separated by spaces (or single end-of-line marks) as if they were words of a single paragraph given as arguments to the `\patterns` primitive command.

[5] Unless you have put several discretionary `\-` breaks in the same word.

[6] The special sign "." marks the beginning or end of a word.

Such a command may be processed only by the initialization version of TEX and is illegal while using (LA)TEX in its normal operating version. The order of patterns in the pattern list should not have any influence, but the list can be maintained much more easily if it is alphabetized on letters only (i.e. disregarding digits).

When extended characters are used, the patterns may contain control sequences, but these are restricted to those that expand to single characters. If you use the technique outlined in Beccari [3], you will have no problems, even if you use the standard (LA)TEX accent macros; just remember to put a digit (possibly `0`) after the control sequences that map to standard characters — i.e. `\oe`, `\ae`, `\aa`, `\o`, `\ss`, and ł.

6 Hyphenation exception lists

TEX can do an excellent job hyphenating words in a particular language but it is not perfect, because every language uses words borrowed from other languages or created with foreign roots and local endings. In some instances, especially when patterns are generated with `patgen`, unusual words may have been omitted from the list fed to `patgen` so that these words might get hyphenated incorrectly.

These exceptions can be addressed by using "hyphenation exception lists", one for each language, that consists of space-delimited hyphenated words given as arguments to the `\hyphenation` primitive. Originally Knuth prepared a list of 14 exceptions for English, but several years of usage have produced a continuously growing list of exceptions that is maintained in the CTAN archives. On this subject it is interesting to read the words by B. Beeton that accompany the 1992 US English exception log [4].

We are of the opinion that exception lists should be avoided as much as possible, at least for those languages with patterns made by hand: if manual creation was possible, it means that the syllabification rules were simple enough to consider most, if not all, normal situations. Hyphenation exception lists then become useful only in specific documents where unusual words are used: — first and/or family names, foreign toponyms, chemical compound names, and the like. The one exception to this seems to be English: while the number of hyphenation rules would seem to argue that the manual approach could not be considered reasonable, the language is widely used and a manually-prepared hyphenation exception log is regularly maintained and updated.

In most other languages hyphenation exception lists could become too difficult to create, especially if such languages are flexive.[7] With such languages — and all Romance languages belong to this class — the conjugation of a verb might include from 60 to 80 different forms, so that if an exception involves the stem of a verb, some 60 to 80 different entries must be made in the `\hyphenation` list for that one verb.

As a concluding remark it is worth noting that the patterns inserted via the `\patterns` command are static: once they have been processed by `initex`, you cannot change them, unless you create a new format. Hyphenation exceptions introduced via the `\hyphenation` command are dynamic, and you can add new exceptions at any point in your compuscript. Any new exception gets added to the list valid for the current language, and if a word is entered twice (supposedly with different hyphen points) the last one is the one TEX uses. Hyphenation exceptions are global and are not limited by group delimiters.

7 TEX hyphenation mechanism

TEX's hyphenation mechanism consists in examining each word to find if it appears in the `\hyphenation` argument of the current language; if so, it hyphenates the word accordingly; otherwise, it examines all possible patterns for dividing the word, patterns that must appear in the pattern list for the current language. This done, it compares and saves the highest digits that the several patterns have produced between the same pairs of letters, and inserts (implicit) discretionary breaks where odd digits appear.

To tell the whole truth, TEX actually does not insert such discretionary breaks before a certain number of letters from the beginning of the word and after another (possibly different) number of letters from the end; with version 3.xx of TEX, such numbers, call them λ and ρ respectively, can be set with the commands

$$\texttt{\lefthyphenmin} = \lambda$$
$$\texttt{\righthyphenmin} = \rho$$

With US English, the default values are $\lambda = 2$ and $\rho = 3$, but for UK English, the left limit is $\lambda = 3$; for Italian, where there are no mute vowels, it is correct to put $\lambda = 2$ and $\rho = 2$, although, if the line width is sufficiently large, $\lambda = 3$ and $\rho = 3$ is more elegant. These two numbers, therefore, relate to the

[7] 'Flexive' describes languages where nouns, adjectives and verbs have different forms (or spellings, if you wish) to show additional elements of meaning: singular or plural, masculine or feminine, case (nominative, accusative, etc., as in Latin) for nouns and/or adjectives, and the many forms due to verb conjugation.

typographic style, not to the hyphenation mechanism, and may be varied according to individual language/national typographical practice and to the particular style one prefers. This implies that patterns and hyphenation exceptions lists should not consider these two values and should produce correct breakpoints even if they are set at $\lambda = 1$ and $\rho = 1$.

8 Tools

Constructing a pattern list for a language whose hyphenation rules are not too complicated is not a difficult task; you just have to organize yourself with the following "tools":

1. a grammar or a very good personal knowledge of the language;
2. a dictionary where syllabification is shown;[8]
3. any national or language regulations, if they exist, that establish rules for typographical hyphenation;
4. if available, an excellent tool would be an orthographic dictionary in computerized form; for Italian we found the one prepared by Luigi Bianchi (to be used with `AMSpell` [6]);
5. a good handbook for typographical typesetting practice.

Another couple of tools are the `\showhyphens` macro provided by TEX, and a macro set provided by Eijkhout [8], called `\printhyphens`.

The first tool, `\showhyphens`, outputs TEX hyphenation on the screen and into the `.log` file in the form of the usual warning message for underfull hboxes. Eijkhout's macros instruct TEX to set one word per line with hyphens inserted at the breakpoints identified by TEX's hyphenation algorithm. The second approach is more elegant, and allows you to produce a clean hardcopy; its only drawback is that fragile commands tend to break up. `\showhyphens` is much simpler, but its output (from the `.log` file) does not contain the extended characters; on the other hand, the output may contain strange symbols[9] or "double caret" sequences, so that its interpretation requires a little skill by the user.

Then you should translate the "spoken" syllabification rules into "abstract" statements of the form:

> if v_i and c_i are vowels and consonants respectively, hyphenate as follows:
>
> - $v_1c_1v_2 \to v_1\text{-}c_1v_2$
> - $v_1c_1c_2v_2 \to v_1c_1\text{-}c_2v_2$
> - and so on (these rules are just examples and do not refer to any particular language)

If you succeed in translating all the "spoken" rules in the above form, then you can proceed to construct the pattern list. Otherwise, `patgen` is probably the only practical solution: you must start from the very beginning and construct a huge list (several thousand words) of perfectly spelled and hyphenated words taken from a language dictionary, then process this list by means of `patgen`. But before starting this heavy task be sure to do everything you can in order to find out if someone else has already done it; public archives are there for that purpose!

We will now describe the manual procedure. Something to keep in mind as you translate the spoken rules into abstract form: prepare a list of words that contain examples of application of the rules; counter examples are also precious elements in this list. Both can later be used to easily check the performance of your hyphenation algorithm.

9 Romanian syllabification "spoken" rules

Our colleague Tulei was able to produce a list of Romanian syllabification spoken rules in three forms which are not completely equivalent so that some intelligent interpretation must be introduced:

1. First form [1]:
   (a) if one vowel is followed by a single consonant, the latter belongs to the following syllable;
   (b) two vowels that do not form a diphthong belong to different syllables;
   (c) *i* and *u* between other vowels behave as semi-consonants and start a new syllable;
   (d) if a vowel is followed by several consonants the first consonant belongs to the left syllable and the other consonants to the right syllable with the exception of the following items;
   (e) if one of the consonants *b*, *c*, *d*, *f*, *g*, *p*, *t*, *v* is followed by *l* or *r*, the pair cannot be split;
   (f) the groups *ct*, *cţ*, and *pt* preceded by one or more consonants get split between the first and the second elements of the group;
   (g) prefixed words are separated according to etymology if the component words maintain their integrity.
2. Second form [13]:

[8] This is not a trivial recommendation; except for English, we do not know of any language for which syllabification is a standard feature of every dictionary.

[9] They are the symbols that the computer has in its internal tables for driving the screen or the printer in correspondence with the internal TEX character "numbers".

(a) a single consonant between two vowels belongs to the second syllable;

(b) two consonants between two vowels are separated unless the second is an *l* or an *r*;

(c) three or more consonants between two vowels are divided so as to leave one or two consonants with the second syllable, the latter case occurring when the last consonant is *l* or *r*;

(d) vowels forming a hiatus are divided;

(e) prefixed words are divided according to their etymology.

3. Third form [5]:

(a) vowels forming a hiatus are divided;

(b) a single consonant between two vowels belongs to the second syllable; for the application of this rule the digraphs *ch* and *gh* are considered a single consonant, as are digraphs imported from other languages which are not "regular" in Romanian words, such as, for example, *sh*;

(c) two consonants are divided unless an *l* or an *r* follows one of the consonants *b, c, d, f, g, h, p, t, v*;

(d) three consonants are divided after the first one if the group ends with *l* or *r*; otherwise, they are divided after the second consonant. The "regular" Romanian consonant triplets of the latter group are: *lpt, mpt, mpţ, ncs, nct, ncţ, ndv, rct, rtf, stm*;

(e) four or five consecutive consonants are divided after the first one, except in adapted words and neologisms.

For rule 3e the underlying concept seems to be that division must take place where the second syllable can be pronounced by a normal Romanian speaker. In other words, the second syllable must start with the longest set of consonants that can be found at the beginning of other Romanian words. The correct division is therefore `ang-strom`, not `an-gstrom` because there is no Romanian word starting with *gstr*, but there are plenty starting with *str*.

Another remark: among the grammars examined, none considers the group *ngv* that appears in at least one word that is linguistically important: *lingvistic*. By analogy with the previous remark, since no Romanian word starts with *gv*, we have decided to adopt the division `ng-v`.

The *cratima* or *liniuţă de unire* (intra-word dash or hyphen) is used to connect most compound words, but it is also used to mark vocalic elision, much as French and English use an apostrophe. This is a minor problem because TEX hyphenates words containing the intra-word dash or hyphen only in correspondence with the hyphen character; in the case where the *cratima* is used for marking vocalic elision there should be no line break.

Consonants:	b, c, d, f, g, h, j, k, l, m, n, p, r, s, ş, t, ţ, v, x, z
Vowels:	a, â, ă, e, i, î, o, u
Special letters:	q, w, y

Table 1: Classification of Romanian letters

10 Romanian "abstract" rules

Let us put the Romanian spoken rules into abstract form. As usual, let c_i be the consonants, l_i the liquid consonants *l* or *r*, v_i the vowels in general, o_i the vowels belonging to the set $\{a, â, ă, e, o\}$, and `x` is a specific letter (*x* in this example). This yields the following abstract rules:

$$v_1c_1v_2 \quad \to \quad v_1\texttt{-}c_1v_2 \tag{1}$$

if $c_1 \in \{l_i\}$ and $c_2 \notin \{l_i\}$ then

$$v_1c_2c_1v_2 \quad \to \quad v_1\texttt{-}c_2c_1v_2 \tag{2}$$

$$v_1c_3c_2c_1v_2 \quad \to \quad v_1c_3\texttt{-}c_2c_1v_2 \tag{3}$$

else

$$v_1c_2c_1v_2 \quad \to \quad v_1c_2\texttt{-}c_1v_2 \tag{4}$$

if $c_2c_1 = \texttt{ct}$ or
$c_2c_1 = \texttt{cţ}$ or
$c_2c_1 = \texttt{pt}$ or
$c_2c_1 = \texttt{pţ}$ then

$$v_1c_3c_2c_1v_2 \quad \to \quad v_1c_3c_2\texttt{-}c_1v_2 \tag{5}$$

else

$$v_1c_3c_2c_1v_2 \quad \to \quad v_1c_3\texttt{-}c_2c_1v_2 \tag{6}$$

$$v_1c_4c_3c_2c_1v_2 \quad \to \quad v_1c_4\texttt{-}c_3c_2c_1v_2 \tag{7}$$

$$v_1c_5c_4c_3c_2c_1v_2 \quad \to \quad v_1c_5c_4\texttt{-}c_3c_2c_1v_2 \tag{8}$$

end if end if

if $v_1 = v_2$ then

$$v_1v_2 \quad \to \quad v_1\texttt{-}v_2 \tag{9}$$

else if $v_1 = \texttt{i}$ or $v_1 = \texttt{u}$ then

$$o_1v_1o_2 \quad \to \quad o_1\texttt{-}v_1o_2 \tag{10}$$

end if

In order to apply the above rules it is necessary to define the sets of vowels and consonants; in Romanian we have the situation summarized in Table 1. The letters *q*, *w*, and *y* are classified as special because they appear only in words not strictly Romanian, that is imported or adapted from foreign languages; in order to hyphenate at least some of these imported words, such letters will be included in the patterns. Furthermore the digraphs *ch*, *gh*,

sh, and the like, that are not listed in Table 1, must be counted as a single consonant.

We considered hiati, diphthongs and triphthongs only to a limited extent. Romanian is very rich in diphthongs and triphthongs, but the same couples or triplets of vowels may be indivisible or form a hiatus in different words, or in the conjugation or declination of the same word they might play a different rôle and become divisible when they used not to be — and vice versa. Since it is so difficult to classify pairs or triplets of vowels as diphthongs or triphthongs, the best thing to do is not to divide them at all, except when rules 9–10 are applicable. Furthermore, let us remember the typographic point of view of avoiding line breaks within a vocalic group.

11 Romanian patterns

With Romanian, as with other languages where the rules refer to vowels and consonants, it is necessary to establish if the former or the latter play a more important rôle in hyphenation. If we exclude the division of vocalic clusters (except when rule 10 applies), there is no doubt that consonants are the discriminating elements, so that patterns may be built focusing on consonants.

When we want to apply rules 1–10, we must not implement the patterns by making all the combinations of letters implied by such rules. We would otherwise create a set of patterns that would be unnecessarily large, because it would contain combinations that never occur in the real language.

Furthermore we use the smallest weight-digits available, remembering that `0` is also implicitly used when we do not write anything. So we start with omitting all vowels, unless specifically required, and start producing the following patterns that should take care af all single intervocalic consonants, and all single consonants at the beginning and ending of words:

```
1b 1c 1d 1f 1g 1h 1j 1k 1l 1m 1n 1p 1q 1r
1s 1ş 1t 1ţ 1v 1x 1z
.b2 .c2 .d2 .f2 .g2 .h2 .j2 .k2 .m2 .p2
.s2 .ş .t2 .ţ .v2 .z2
2b. 2c. 2d. 2f. 2g. 2h. 2j. 2k. 2l.
2m. 2n. 2p. 2r. 4s. 2ş. 4t. 2v. 2x.
2z.
```

Now we consider the *l* and *r* rules starting with statement 2; in passing, we also add the digraphs *ch* and *gh*, the digraph *sh* that appears so often in foreign words, and the group *tz* that appears in several adapted technical words:

```
c2h g2h s2h t2z
b2l c2l d2l f2l g2l h2l k2l p2l t2l v2l
b2r c2r d2r f2r g2r h2r k2r p2r t2r v2r
```

Next, we allow word division between other consonants, inhibiting word division before the first one of the pair; as well, we introduce some patterns for dealing with the special letters *w* and *y*:

```
2bc 2bd 2bj 2bm 2bn 2bp 2bs b3s2t 2bt
2bţ 2bv
2cc 2cd 2ck 2cm 2cn 2cs 2ct 2cţ 2cv 2cz
2dg 2dh 2dj 2dk 2dm 2dq 2ds 2dv 2dw
2fn 2fs 2ft
2gd 2gm 2gn 2gt 2g3s2 2gv 2gz
2jm
2hn
2lb 2lc 2ld 2lf 2lg 2lj 2lk 2lm 2ln 2lp
2lq 2lr 2ls 2lt 2lţ 2lv 2lz
2mb 2mf 2mk 2ml 2mn 2mp 2m3s2 2mţ
2nb 2nc 2nd n3d2v 2nf 2ng 2nj 2nl 2nm
2nn 2nq 2nr 2ns n3s2a. n3s2ă n3s2e
n3s2i n3s2o n3s2cr n3s2f ns3h n3s2pl
n3s2pr n3s2t 2nş n3ş2c n3ş2t 2nt 2nţ
2nv 2nz n3z2dr
2pc 2pn 2ps 2pt 2pţ
2rb 2rc 2rd 2rf 2rg 2rh 2rj 2rk 2rl 2rm
2rn 2rp 2rq 2rr 2rs r3s2t 2rş 2rt 2rţ
2rv 2rx 2rz
2sb 2sc 2sd 2sf 2sg 2sj 2sk 2sl 2sm 2sn
2sp 2sq 2sr 2ss 2st 2sv 2sz
2şn
2şt
2tb 2tc 2td 2tf 2tg 2tm 2tn 2tp 2ts 2tt
2tv 2tw
2vn
1w wa2r
2xc 2xm 2xp 2xt
1y 2yb 2yl 2ym 2yn 2yr 2ys
2zb 2zc 2zd 2zf 2zg 2zl 2zm 2zn 2zp 2zr
2zs 2zt 2zv
```

Some of the patterns have a non-repetitive look in order to take care of rules 6, 7 and 8; for example `n3s2t` overrides `2st` so that a word like *monstru* can be hyphenated correctly as `mon-stru`. With just the "regular" pattern `2ns`, the hyphenation would have been `mons-tru`.

Finally, we introduce the patterns that involve vowels; in particular, we inhibit hyphenation when the last syllable (and the whole word altogether) ends with an *i*. This is useful because the final *i* is almost mute in the endings of some masculine non-articulated nouns and adjectives, and in such cases line breaks are not allowed [7]. But since TEX does not know anything about the language, we take

a conservative approach and inhibit division before any ending syllable terminated with an *i*.

```
a1ia a1a a1ia a1ie a1io ă1ie ă1oa â1ia
e1e e1ia i1i i2ii. 2i. o1o o1ua o1uă
u1u u1ia u1al. u1os. u1ism u1ist u1iţt
2bi. 2ci. 2di. 2fi. 2gi. 2li. 2mi.
2ni. 2pi. 2ri. 2si. 2şi. 2ti. 2tri.
2ţi. 2vi. 2zi.
```

Eventually we add a few patterns that work with some prefixes:

```
.ante1 .anti1
.contra1
.de3s2cri
.de2z1aco .de2z1amă .de2z1apro
.de2z1avan .de2z1infec .de2z1ord
.i2n1ad .i2n1am .i2n1of .î2n1ăsp
.î2n1ad .î2n3s2 .în3ş2
.ne1a .ne1î .nema2i3 .ne3s2ta
.re1ac .re1î
.su2b1ord .su2b3r
.supra1
.tran2s1 .tran4s3pl .tran4s3f
```

12 Tests and conclusion

We were able to obtain a small pattern file containing 350 patterns[10] and requiring 31 ops. This pattern set is much simpler than that mentioned in Sojka and Ševeček, and the results are simpler too. However, although the ratio between the number of patterns cited by these two authors and our set is approximately 12, our results are not 12 times poorer. It might be interesting to test them on a large set of Romanian words in order to compare the differences. As for our experience, we used these patterns to typeset a large number of Romanian documents but never experienced an incorrect line break.

Using Eijkhout's `\printhyphens` macro, we can verify what we got: feeding this macro some test words[11] we get the following:

a-do-les-cen-ti-lor	in-dus-triei
a-pro-band	in-e-gal
ba-ia	jert-fă
Bu-cu-resti	mon-stru
con-struc-tor	post-de-cem-bris-tă
dum-ne-zeu	vre-mea
drep-tu-lui	Wa-shing-to-nu-lui
dez-a-van-ta-jul	watt-me-tru
fla-shul	

[10] One pattern not mentioned here is `2-2`, which makes it possible to distinguish between the *cratima* (referred to in section 9) and the hyphen character, by using appropriate category codes.

[11] For testing purposes, we set $\lambda = 1$ and $\rho = 1$; in normal Romanian typesetting, it is better to set $\lambda = 2$ and $\rho = 2$.

Let us point out that with the help of such tools one can verify both the validity of the pattern set and if any patterns are missing. For example, if a word turns out to have incorrect hyphen points, it is possible to find out why: — wrong patterns? missing patterns? wrong weights? — and to proceed with corrections. This is how we discovered that we had originally missed the pattern `2tt`, which refers to a double consonant not present in "normal" Romanian; in fact, without this pattern, *wattmetru* gets hyphenated as `wa-tt-me-tru`, which is obviously wrong since it has a word fragment without vowels. Similar situations can be easily spotted by means of the word list that had been prepared together with the patterns, as suggested above. With the help of a dictionary it is possible to spot unusual or doubtful words, include them in the test word list, and find out if they get hyphenated correctly.

A few concluding remarks. Romanian is not particularly easy nor particularly difficult to hyphenate. If one gives up the possibility of TEX doing the whole task of separating the prefixes, and is willing to using `"-` or similar macros for inserting discretionary soft breaks in the prefixed words, the pattern file one gets is of very modest size although it produces correct hyphenation in most circumstances. Such a small file is compatible with the memory limitations of small implementations of TEX the program; (LA)TEX can be initialized with several hyphenation patterns at the same time, allowing the user to create a multilanguage tool that allows him/her to typeset texts in several languages without the need for changing software when passing from one language to another.

Acknowledgments

We would like to thank the precious cooperation of Mihai Lazarescu and Janetta Mereuta who helped very much with the queries and discussions on the Internet and with the creation of a test word list with normal and unusual Romanian words.

References

[1] A.V., *Îndreptar ortografic, ortoepic şi de punctuaţie*, Bucureşti, 1971 (ed. III-a).

[2] Beccari C., "Computer Aided Hyphenation for Italian and Modern Latin", *TUGboat* 13(1):23–33, April 1992.

[3] Beccari C., "Configuring TEX or LATEX for typesetting in several languages", *TUGboat* 16(1):18–30, March 1995.

[4] Beeton B., "Hyphenation Exception Log", *TUGboat* 13(4):452–457, December 1992.

[5] Beldescu G., *Ortografia actuala a limbii române*, Bucureşti: Editura Ştiinţifică şi Enciclopedica, 1985.

[6] Bianchi L., `italian.zip`. [Includes the complete set of tools for checking correct Italian spelling in ASCII and (LA)TEX compuscripts; available from `ftp.yorku.ca` or contacting directly L. Bianchi at `lbianchi@sol.yorku.ca`.]

[7] Breban V., Bojan M., Comşulea E., Negomineanu D., Şerban V., Teiuş S., *Limba română corectă*, Bucureşti: Editura Ştiinţifică, 1973.

[8] Eijkhout V., "The bag of tricks", *TUGboat* 14(4):424, December 1993.

[9] Ferguson M., `frhyph.tex`, `frhyph7.tex`, `frhyph8.tex`. [Hyphenation pattern files available from the CTAN archives in the directory `/tex-archive/languages/french`.]

[10] Knuth D.E., *The TEXbook*, Reading Mass.: Addison-Wesley, 1990.

[11] Pusztai A. and Ardelean Gh., *LATEX, Ghid de utilizare*, Bucureşti: Editura Tehnică, 1994.

[12] Sojka P., Ševeček P., "Hyphenation in TEX — Quo Vadis?", *Proceedings of the Eighth European TEX Conference* (Sept. 26–30, 1994, Gdańsk, Poland), 59–68.

[13] Zamsa E., *Limba română, recapitulari şi exerciţii*, Bucureşti: Editura Ştiinţifică, 1991.

Appendix

Useful shorthand macros

Although Table 1 shows that just five diacriticized characters appear in the Romanian alphabet, the language makes frequent use of these characters, so that the ASCII source `.tex` file is filled with sequences such as `\u{a}`, `\^a`, `\^{\i}`, `\c{s}`, `\c{t}` (and their uppercase counterparts), to the point where the source file is almost unreadable. A single word may contain several such characters, as for example *ţîşnitură*, `\c{t}\^{\i}\c{s}nitur\u{a}`, that contains four of them, so that to key in the source file is quite error prone, and finding and correcting possible errors turns out to be quite difficult.

If you have a Romanian keyboard, you can easily map the input Romanian characters (with internal code higher than 127) to the appropriate sequences: on the screen the text appears without backslashes and braces, but if you have to send your file to somebody else on some computer network global substitutions have to be made to put back the cumbersome sequences.

Another approach would be to make some character active and then define it in a suitable way, so that you may read your source file with a minimum of extra characters interspersed.

German and Dutch users, who have similar problems, have chosen the double quote `"`; nothing prevents us from doing the same for Romanian, but the double quote is used so many times explicitly or behind the scenes by TEX, especially for representing internal codes in hexadecimal notation, that we prefer another, more "innocent" character — the exclamation mark `!`.

A LATEX guide has been published recently in Romania [11], where the apostrophe sign has been made active and defined in such a way as to save a lot of keystrokes while keying in the source text. We wonder if the authors actually used the macros they suggest on page 109 of their guide because the `\newcommand` command cannot (ordinarily) operate on active caracters, but only on control words — they probably used a regular `\def` command. Several other TEX users around Romania[12] who answered our questions on the Internet revealed that they are using definitions similar to the ones published in the Romanian LATEX guide. But the same criticism remains: the apostrophe is used internally by the plain and LATEX format files for identifying octal numbers, and it becomes really difficult to create correct definitions so that character "activity" does not interfere with octal notation.

Therefore we will stick with our choice of the exclamation mark, but what we state here can be applied to any other "innocent" character. If you use babel, then you should add the exclamation mark to the special TEX characters and provide an argument to `\extrasromanian` such as, for example:

```
\addto\extrasromanian{%
  \babel@add@special\!}
\addto\extrasromanian{%
  \babel@remove@special\!}
\addto\extrasromanian{%
  \babel@savevariable{\catcode`\!}%
  \catcode`\!\active}
```

In the style file there must be a statement that memorizes the meaning of `!` before changing catcode:

```
\let\@xcl@@=!
```

and a set of definitions for the active exclamation mark:

```
\begingroup
\catcode`\!=13
\gdef!{\protect\p@xcl@}
\endgroup
```

[12] They are too numerous to be cited here, but we take this opportunity to thank them warmly.

Some macros must be defined in order to test if what follows the exclamation mark is a non-expandable token (i.e. a character token or a primitive TEX command), or has an expansion:

```
\def\@Macro@Meaning#1->#2?{%
              \def\@Expansion{#2}}
%
\def\TestMacro#1#2#3{%
    \expandafter
      \@Macro@Meaning\meaning#1->?%
    \ifx\@Expansion\empty
      \def\@Action{#3}%
    \else
      \def\@Action{#2}%
    \fi
    \@Action}
```

More definitions are needed, and the apparently tortuous path followed before arriving at the desired goal is due to the necessity of making such macros robust, so that they do not break apart when they appear in moving arguments, such as when writing to auxiliary files for cross-reference purposes or for preparing indices or tables of contents.

```
\def\p@xcl@{\ifmmode
              \@xcl@@
           \else
              \expandafter\ExCl
           \fi}
%
\def\ExCl{\futurelet\eXcl\exCl}
%
\def\excL{\Excl{}}
%
\def\exCl{\TestMacro\eXcl{\excL}{%
           \ifcat\eXcl a%
              \let\eXcL\Excl
           \else
              \ifcat\eXcl "%
                 \let\eXcL\Excl
              \else
                 \let\eXcL\excL
              \fi
           \fi
           \eXcL}}
```

Finally the definitions we were aiming at:

```
\def\Excl#1{\expandafter
\ifx\csname @xcl@#1\endcsname \relax
    \@xcl@@\space #1%
\else
    \csname @xcl@#1\endcsname
\fi}
%
\def\hz{\nobreak\hskip\z@}
```

!a	ă	!A	Ă
!i	î	!I	Î
!s	ş	!S	Ş
!t	ţ	!T	Ţ
!␣	!	!"	`
!-	-	!|	

Table 2: Sequences obtained with the active exclamation mark and their results. In math mode the exclamation mark preserves its meaning.

```
%
\expandafter
  \def\csname @xcl@a\endcsname{\u{a}}
\expandafter
  \def\csname @xcl@A\endcsname{\u{A}}
\expandafter
  \def\csname @xcl@s\endcsname{\c{s}}
\expandafter
  \def\csname @xcl@S\endcsname{\c{S}}
\expandafter
  \def\csname @xcl@t\endcsname{\c{t}}
\expandafter
  \def\csname @xcl@T\endcsname{\c{T}}
\expandafter
  \def\csname @xcl@i\endcsname{\^{\i}}
\expandafter
  \def\csname @xcl@I\endcsname{\^I}
\expandafter
  \def\csname @xcl@"\endcsname{^^12}
\expandafter
  \def\csname @xcl@-\endcsname{\char"7F}
\expandafter
  \def\csname @xcl@|\endcsname{\hz\-\hz}
```

When you select the Romanian language the exclamation mark becomes active and you get the shorthand notations summarized in Table 2. The last line of this table requires some additional comment.

1. The sequence `!|` produces a discretionary break that does not inhibit hyphenation in the rest of the word, and is useful for marking the boundary between a prefix and a word stem. This sequence complements the regular TEX sequence `\-` that inserts a discretionary break but inhibits hyphenation in the rest of the word.
2. The sequence `!-` produces the short dash that is used for connecting compound words (quite rare in Romanian); it inserts the `\hyphenchar` that TEX treats in a special way for what concerns line breaking. In fact, when this character appears, TEX breaks the line, if necessary, only after the short dash.

3. The frequent *cratima*, i.e. the short dash to be used in place of an elided vowel or for connecting enclitic pronouns (across which line breaks are forbidden), is obtained by the usual `-` sign, but to avoid having TEX treat it as the `hyphenchar`, it is necessary (a) to chose a different entry in the font table for the hyphen character, and (b) to assign the "minus sign" an `\lccode` different from 0, so that in text mode it can be treated as a regular letter and it is possible to enter a special pattern, `2-2` in the pattern list, so that line breaks are forbidden across it. By so doing, the number of ops increases by 1, but still remains very reasonable.

   The extended fonts actually include two short dashes, one with hexadecimal code `"2D`, and the other with hexadecimal code `"7F`. For Romanian, it is therefore necessary to assign a lowercase code to `"2D` so as to use it as a *cratima*, and to declare `"7F` to be the `\hyphenchar` for the current extended font.

   In order to revert to the original situation when a different language is selected, it is necessary to restore the `\lccode`s and the `\hyphenchar` assignments. We did not investigate how this is done in babel, but we did it with our implementation, which is less general, compared with the facilities offered by babel.

It is worth noting that the exclamation mark followed by a character different from one of those listed in Table 2 reproduces itself; since this mark is commonly used at the end of a sentence, and therefore is followed by a space or an end-of-line mark, a space is inserted by default by the macro expansion.

A sample (source) text follows, so as to appreciate the shorthand notations just introduced.

```
Problema desp!ar!tirii cuvintelor !in
silabe se pune mai ales !in scriere,
unde se ivesc dificult!a!ti c\^and
este vorba de vocale sau de consoane
succesive.  Dar problema desp!ar!tirii
!in silabe este str\^ans legat!a !si de
pro!|nun!tarea corect!a a unor cuvinte
care con!tin diftongi, triftongi sau
vocale in hiat, dup!a cum se va vedea
mai departe.

La categoria de nume geografice
teritorial!-administrative, {\em
!Indreptarul ortografic} prevede c!a
acestea !"se scriu cu ini!tial!a
majuscul!a la toate cuvintele
componente``, preciz\^andu-se !in
acela!si timp c!a: !"Numele generice
{\em deal, fluviu, insul!a, lac, munte,
peninsul!a, r\^au, vale} etc., c\^and
nu fac parte din denumire, se scriu cu
ini!tial!a mic!a``.
```

Note that *â* still requires the full accent sequence,[13] but it is just three keystrokes, compared with the numerous keystrokes required by the other special characters. Notice the sequence `!-` that connects a compound word (after which TEX will break the line if necessary) and the *cratima* that connects the enclitic pronoun (where hyphenation is forbidden). And finally, notice the reversed and lowered quotation marks and the soft discretionary inserted after the prefix *pro*.[14] The corresponding typeset text appears as such:

> Problema despărţirii cuvintelor în silabe se pune mai ales în scriere, unde se ivesc dificultăţi când este vorba de vocale sau de consoane succesive. Dar problema despărţirii în silabe este strâns legată şi de pronunţarea corectă a unor cuvinte care conţin diftongi, triftongi sau vocale in hiat, după cum se va vedea mai departe.
>
> La categoria de nume geografice teritorial-administrative, *Îndreptarul ortografic* prevede că acestea `se scriu cu iniţială majusculă la toate cuvintele componente", precizându-se în acelaşi timp că: `Numele generice *deal, fluviu, insulă, lac, munte, peninsulă, râu, vale* etc., când nu fac parte din denumire, se scriu cu iniţială mică".

◇ Claudio Beccari
Dipartimento di Elettronica
Politecnico di Torino
Turin, Italy
Email: beccari@polito.it

◇ Radu Oprea
Dipartimento di Elettronica
Politecnico di Torino
Turin, Italy

◇ Elena Tulei
Universitatea Tehnică de Construcţii
Bucureşti, România

[13] Up to a couple of years ago, *â* was used only in the word *România* and its derivatives. Since the last spelling reform this sign is used more often; in the example we have used the modern spelling.

[14] Actually there is no need for this discretionary break — it was put there just to show its use. With our patterns we did not succeed in finding a sample text containing something that really requires the use of a soft discretionary break.

TEX and Linguistics

Christina Thiele

Over the past few years a small but persevering group of us have been working on gathering information about the use of TEX for typesetting linguistics material. And for more than just the past few years, many people have been developing macros to deal with the specialized formatting which often arises in this field.

The information-gathering has yielded some useful results which have been announced and published in various places; this short piece simply collects that all into one article.

There are three main items to present here: the Technical Working Group (TWG), the `ling-tex` list, and an inventory of style files (`ling-mac.tex`).

1 The TWG for TEX and Linguistics

This TWG was formed in July of 1994, following the previous half-year's activity on the `ling-tex list` (see below). While activity in the TWG has been quite minimal, the following people are now part of the group: Christopher Manning (Stanford University), Rei Fukui (University of Tokyo), myself as chair, and Stuart Shieber (Harvard University). The TWG maintains the inventory of macros (see below), and is planning a similar inventory of fonts.

The TWG is officially designated as WG-94-10 (SI-TWG = special interest). Our mandate currently reads as follows: "The main goal is to study and discuss the requirements for typesetting linguistics in TEX and as a means of identifying, examining, testing, and comparing macros, fonts, style files and other aids for typesetting linguistics." The liaison to the Technical Council is Yannis Haralambous.[1]

2 The `ling-tex` list

The purpose of `ling-tex` is to identify material which is available to typeset linguistics text with TEX, and to also stimulate testing, improvements to code and documentation, and so on, all with as much cooperation and assistance from the original authors as possible. It is no coincidence that this is very similar to the mandate of the TWG: at the time the Technical Council was inviting people to consider starting up special interest working groups, it wasn't clear to me how much of an interest there might be in TEX and linguistics. The list was a way to gauge that interest. Initial response was such that by the end of its first month over 150 subscribers had joined. There was — and continues to be — interest: the list currently has some 250 subscribers.

Begun late in 1993, the list was originally set up by George Greenwade at SHSU. `ling-tex` has now moved to the University of Oslo, where it is maintained by Dag Langmyhr. To subscribe to the list, send a message to `ling-tex-requests@ifi.uio.no`. To post messages, address them to `ling-tex@ifi.uio.no`. A Web page has recently been started and will be ready for public viewing by the time you read this article. The address is `http://www.ifi.uio.no/~dag/ling-tex.html`.

3 The `ling-mac.tex` inventory

One immediate result of the `ling-tex` list was information about style files various people had developed to deal with such formatting issues as tree diagrams, examples, glosses, and attribute-value matrices, as well as bibliographies via BIBTEX files.[2] The first draft was circulated in early 1994; it is now regularly updated and posted to the `ling-tex` list. The inventory appears below; most material is for LATEX rather than plain.

TEX and LATEX Macros for Linguistics

The following list is not intended to be exhaustive or complete; it is based on information which has come to light as people have posted messages to the `ling-tex` mailing list.

The material is all public domain, but the usual requests for citing authorship, not changing the contents without changing the file name, and so on apply. These are the results of volunteers efforts, and a desire to share those efforts with others; this should always be kept in mind. Constructive criticism, helpful suggestions, or offers of revised coding or wording are always welcome.

Index of `.sty` files

1. avm-doc.tex, avm.sty
2. cgloss4e.sty
3. chomsky.sty
4. cjl-glosses.tex
5. cm-lingmacros.sty
6. covingtn.tex covingtn.sty
7. french.sty
8. glex.sty
9. gloss.tex, gloss.doc.
10. lingmacros.sty
11. lsalike.sty, lsalike.bst

[1] For a complete list of Technical Working Groups, ftp the file `twg-list.tex` from CTAN in `tex-archive/usergrps` (`.dvi` and `.ps` files are also available.

[2] The `ling-mac.tex` file can be found on CTAN in `tex-archive/info/ling-mac.tex`. Contact the TWG for additions and corrections.

12. numquote.doc, numquote.tex, enum.sty
13. pstrees
14. pstricks
15. tree-dvips
16. voorbeeldom.sty

Details on various `.sty` files

1. **avm-doc.tex, avm.sty:**
   Christopher Manning
   (`manning@csli.stanford.edu`).
   Macros for attribute-value matrices. Documentation available (but not printed in this collection).
   `csli.stanford.edu:pub/TeXfiles`

2. **cgloss4e.sty:**
   This is a modified version of `covingtn.sty` by Hap Kolb and Craig Thiersch. For glosses; as with `covingtn.sty`, does not require ampersands (`&`) to align sets of glossed items.
   "[The f]ollowing borrows from M. Covington's style files inspired by Midnight by M. de Groot, adapted to be used with `gbt3.sty`: examples beginning with `\ex` can contain glosses directly. Default is *Linguistic Inquiry* style with all lines in `\rm`."
   No documentation, but file is heavily commented. Posted to `ling-tex`; not currently available on archives.

3. **chomsky.sty:**
   Michael Barr.
   No documentation; however, file is heavily annotated. Some draft documentation by Ch. Thiele

4. **cjl-glosses.tex:**
   Maintained by by Ch. Thiele
   (`cthiele@ccs.carleton.ca`).
   Macros for glosses (seems to work in both plain TEX and in LATEX). Variants for centred, flush right or left glosses, and others. Some documentation; needs testing before it can be put out on the archives.
   Posted to `ling-tex` list; not currently available on archives.

5. **cm-lingmacros.sty:**
   Christopher Manning and Avery Andrews
   (`Avery.Andrews@anu.edu.au`).
   Modified version of `lingmacros.sty` (see below).
   `csli.stanford.edu:pub/TeXfiles`

6. **covingtn.tex, covingtn.sty:**
   Michael Covington.
   LATEX macros for numbered examples, glosses, phrase structure rules, feature structures, discourse representation structures, exercises, reference lists, and miscellany. Documentation.
   CTAN: `tex-archive/macros/latex/contrib/covington`

7. **French Style Files:**
   Bernard Gaulle (`gaulle@idris.fr`).
   French-based style files offering an easy-to-use multilingual scheme to work with other languages (English and German are currently offered). French patterns are up-to-date and there are a lot of test files. This package also offers a way to change your keyboard "on the fly" and to set your default at `initex` time, i.e. when creating your format. Two versions are released per year.
   `ftp.univ-rennes1.fr:pub/GUTenberg/french`

8. **glex.sty:**
   Rob Norris; notes from Chet Creider.
   LATEX macros for numbered glosses. All three lines of a gloss are input; by contrast, `cjl-glosses.tex` only takes care of the first 2 lines, requiring the 3rd line, the translation, to be formatted independently. On the other hand, `glex.sty` works with tabs, while `cjl-glosses.tex` groups each set of word-1 over gloss-1 within braces.
   [Availability not yet determined.]

9. **gloss.tex, gloss.doc:**
   Part of the Midnight Macros set by Marcel van der Goot (`marcel@cs.caltech.edu`).
   Macros for vertically aligning words in consecutive sentences. Documentation.
   CTAN: `tex-archive/macros/macros/generic/midnight`

10. **lingmacros.sty:**
    Emma Pease, CSLI, Stanford.
    Macros for numbered examples, trees, AVM structures.
    `csli.stanford.edu:pub/TeXfiles`

11. **lsalike.sty, lsalike.bst:**
    Daniel S. Jurafsky, UC Berkeley.
    "*lsalike* style file for BIBTEX. It implements a bibliography format which is very close to the LSA style sheet and resembles the journal *Language*. Among its advantages are that it does the lovely dashed lines for repeated bib entries that makes *Language* bibliographies so easy to read, and it also makes citations of the form 'Chomsky (1965:134)' very easy."
    `ftp.icsi.berkeley.edu:pub/ai/jurafsky`

12. **numquote.doc, numquote.tex, enum.sty:**
    Bob Mercer, U. of Western Ontario.
    LATEX macros for automatic numbering of examples. Documentation.
    Posted to `ling-tex` list; not currently available on archives.

13. **pstrees:**
Avery Andrews; requires `tree-dvips` (see below).
"This package consists of a preprocessor and some macro-definitions, by which linguistics-style trees can be specified as convenient indented lists, with spacing and line-drawing done automatically."
`csli.stanford.edu:pub/TeXfiles`
[Note: `pstrees` contains files which do *not* include the prefix "pstrees"; rather, the main files are `trees.**`, which may cause confusion if one is not careful – Ch.]

14. **pstricks:**
Timothy Van Zandt (`tvz@princeton.edu`).
This is an extensive collection of PostScript macros that is compatible with most TEX macro packages, including Plain TEX, LATEX, $\mathcal{AMS}$-TEX and $\mathcal{AMS}$-LATEX. Included are macros for color, graphics, rotation, trees and overlays. "PSTricks puts the icing (PostScript) on your cake (TEX)!" Documentation.
CTAN: `tex-archive/graphics/pstricks`

15. **qtree:**
Alexis Dimitriadis
(`alexis@babel.ling.upenn.edu`).
Tree macros (written by Jeff Siskind) with a front end allowing trees to be specified in bracket notation. These macros take into account the size of the node labels when designing the tree; it is usually only necessary to give the bracketed structure to obtain beautiful trees. The node labels themselves can be arbitrarily complicated. The output is `.dvi` (not PostScript) directives, thus it can be previewed with xdvi.
Version 2 simplifies the labeling of non-termin nodes and provides triangular "roofs". Documentation is included in the distribution file. [Note: This is a new release, dating from February 1995.]
`ai.uga.edu:/pub/tex`

16. **tree-dvips:**
Emma Pease (`emma@csli.stanford.edu`).
CSLI PostScript drawing macros. These macros were originally created to draw the lines between nodes in the trees created by the tree macros in `lingmacros.sty`. They will only work with dvips version 541 or later (by Tomas Rokicki available on `labrea.stanford.edu`) but can be easily modified to be used with earlier versions of dvips and slightly less easily modified for other `.dvi`-to-PostScript convertors. Documentation. [Formerly: `tree.tex`.]
`csli.stanford.edu:pub/TeXfiles`

17. **treetex:**
Anne Brueggemann-Klein and Derick Wood.
Extensive tree-drawing macro set. Documentation available. See also A. Brueggeman-Klein and Derick Wood, "Drawing trees nicely with TEX," *Electronic Publishing* 2.2:101–115 (1989).
[Availability unknown.]

18. **voorbeeldom.sty:**
Werenfried Spit
(`spit@vm.ci.uv.es`, `spit@ific.uv.es`).
LATEX document-style option which defines an `enumerate`-like environment for typesetting linguistic examples. No documentation, but the `.sty` file has commented examples.
CTAN: `tex-archive/macros/latex/contrib/misc/`

What's available where

CTAN: Most TEXware is available via ftp from CTAN (Comprehensive TEX Archive Network) sites, in the directory `/tex-archive`. CTAN sites include:

United Kingdon	`ftp.tex.ac.uk`
Huntsville, Texas	`ftp.shsu.edu`
Germany	`ftp.dante.de`

The CTAN holdings are too numerous to list here. For more information, get the `readme` files from the `/tex-archive` directory.

CSLI: In addition to CTAN, there has been a longstanding ftp site at Stanford:
`csli.stanford.edu:pub/TeXfiles`

U of Georgia: There is a smaller archive at the University of Georgia which contains files of local interest.
`ai.uga.edu:pub/tex`

U of Tokyo: There is a small archive at the University of Tokyo which contains the latest version of the phonetic font TSIPA and other related files.
`tooyoo.l.u-tokyo.ac.jp:pub/TeX/tsipa`

◇ Christina Thiele
15 Wiltshire Circle
Nepean, Ontario
K2J 4K9 Canada
Email: `cthiele@ccs.carleton.ca`

Font Forum

Introducing METAPOST

Alan Hoenig

I am pleased that the article by Yannis Haralambous which immediately follows these comments is available. People using METAFONT should find it of great interest and significance.

It has long been clear that METAFONT things are of interest to only a limited subset of TEX or LATEX users. Who, after all, has the time to design fonts? The event that I wish now to report should significantly alter this perception. As of April of 1995, John Hobby's METAPOST program has been placed in the public domain, and I'd like to comment on this turn of events.

METAPOST is similar to the METAFONT language, so METAPOST input files look a lot like METAFONT files. However, the output is different — METAPOST produces PostScript output rather than generic font files, so it is printable on any PostScript device. It's not really possible to produce fonts with METAPOST. Its *raison d'être* is really toward the production of high quality graphics for inclusion in a TEX or LATEX document.

There are differences between METAFONT and METAPOST, and these are necessitated by the differences in use and environment. PostScript descriptions are to be device independent, so all of METAFONT's pixel-handling constructs have been removed. Things like the sharp convention and commands like `cullit` are not part of the METAPOST language. Certain enhancements have been added to the METAPOST language in aid of fine graphics.

For example, METAPOST is also able to easily include TEX or LATEX text within its graphic output, so tags on graphs and drawings can now match the text font exactly. The METAPOST package includes special macro packages for drawing graphs and for drawing boxes and ovals. The silly graph in figure 1 (everything above the caption) was produced entirely by METAPOST—it read the data points, connected them with a smooth curve, prepared the coordinate axes, and integrated the LATEX labels and tags all by itself.

The output of a successful METAPOST run is a sequence of files with names like `foo.1`, `foo.2`, and so on. With the `epsf` macro package they are easily included in any document that is post-processed by Tom Rokicki's `dvips` (although it may be that other post-processors can also incorporate them as well). Plain TEX authors include lines like

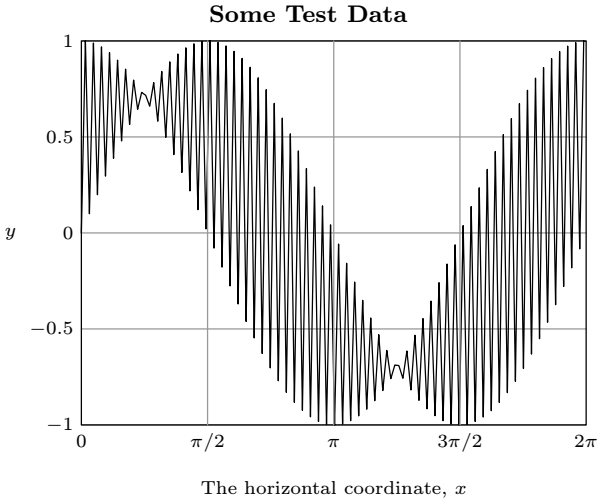


Figure 1: A sample METAPOST graphic.

```
\input epsf
 ...
\epsfbox{foo.1}
```

in their document, while LATEX users say something like

```
\usepackage{epsf}
 ...
\begin{center}
  \leavevmode\epsfbox{foo.1}
\end{center}
```

in their documents. (You may use `\noindent` in place of `\leavevmode` if you wish. These are often, but not always, needed in LATEX environments.)

METAPOST is available from any CTAN archive in the path `tex-archive/systems`. At least two executables already exist, for DOS and for OS/2, and (I believe) it is now part of the Unix `web2c` kit. To learn how to use METAPOST, consult the original *METAFONTbook* (usable by virtue of the many similarities between the two languages), and also the two technical reports by John Hobby. These two reports, numbered 162 and 164, are part of the package, but can be obtained by sending email `send 162` or `send 164` to `netlib@research.com`. These reports are valuable not only for their explications of METAPOST but for the alternative perspectives they also provide for METAFONT.

◇ Alan Hoenig
CUNY, 17 Bay Ave.,
Huntington, NY 11743
Email: `ajhjj@cunyvm.cuny.edu`

Some METAFONT Techniques

Yannis Haralambous

Abstract

This paper presents a few ideas on how to solve certain geometrical problems arising very often in character design, not directly solvable by METAFONT's `plain` macros. The first part of the paper presents two geometrical problems: the "*k problem*" and the "*x problem*", their solutions using dichotomy, and a different solution using path intersections. The latter was proposed earlier on the net by the author; although geometrically correct, it does *not* work in real-world METAFONT practice: a nice example of METAFONT code ... to avoid.

The second part of the paper presents two simple macros for drawing "loose" Bézier curves; in a sense, the opposite of the `tension` operator. Finally, the third part solves a problem stated by Alan Hoenig: how to extract text and data from a METAFONT run, without using the log file. This is done in a straightforward manner by running a Flex-generated preprocessor over the GF file: the Flex code for this utility is given in appendix B.

– – ∗ – –

1 Two geometrical problems, solved by iterated calculations

1.1 Description

Suppose you want to design a character 'K', as in the left part of fig. 1. The character should fit inside a box of width w and height h, and should consist of three strokes: the vertical stroke $z_0 - -z_{0'}$, and the two oblique strokes $z_1 - -z_2$ and $z_{1'} - -z_{2'}$. Only constraint: the point $z_{1l} = z_{1'r}$ should be fixed (for example, its coordinates can be $(0, \frac{h}{2})$). So, here is the problem:

Find a stroke $z_1 - -z_2$ with fixed z_{1l}, y_{2l}, x_{2r}.

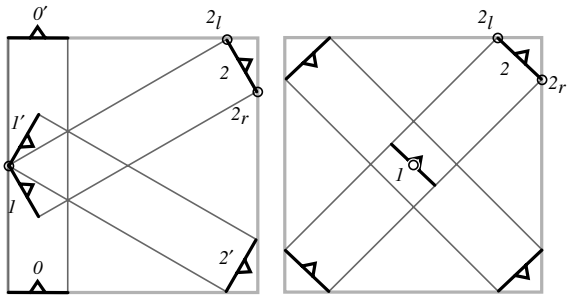


Figure 1: The two problems.

This problem is not trivial, because METAFONT cannot compute pen positions without knowing in advance the angle of the pen (this stands both for defining a new pen with command *pickup pen* and for defining a simulated pen with command *penpos*). Because it arises when designing the letter 'K', we will call this problem the "*k problem*".

The next problem is encountered when designing an 'X', as in the right part of fig. 1. Suppose you want to draw this letter. Once again it should fit inside the box, and should consist of two strokes. To keep the same notation as in the previous case, we have only given names to the pen positions concerning the upper right part of the letter. In this case the constraints are: z_1 is fixed (and not z_{1l} as in the previous case), as well as y_{2l} and x_{2r}. Here again is the problem which we call the "*x problem*":

Find a stroke $z_1 - -z_2$ with fixed z_1, y_{2l}, x_{2r}.

Well understood, in both cases the direction of the stroke must be perpendicular to the angle of the pen: all strokes must keep the same width.

1.2 The solutions (which work)

Let's start with the "*k problem*". In fig. 2, the reader has a closer look at the situation. Point A is fixed, point B must lie on a fixed horizontal line H, and C on a fixed vertical line V. The angle $\widehat{ABC}$ must stay orthogonal. Also the length $\overline{BC}$ of the vector BC is fixed. METAFONT cannot compute the angle ϕ directly, so that it fits to these constraints. But once we have chosen a point B', METAFONT can calculate the corresponding C' so that $AB' \perp B'C'$ and $\overline{B'C'} = \overline{BC}$.

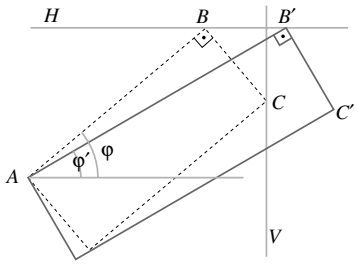


Figure 2: A closer look at the "*k problem*".

So let's choose a random point $B' \in H$, and find the corresponding C'. If C' lies on the right of V then we know that we should move B' more to the left, if C' lies on the left of V then B' should be moved more to the right. We will modify the position of B' by a certain step and start again. This procedure will be iterated until we get close enough to V. The step will be halved every time: because of the well-known equality $\sum_{i \geq 1} \frac{1}{2^i} = 1$ we are sure that this process of iterations will converge to the

correct result. One may argue that the result will always be approximate; this is true in mathematics but a useless remark in computer calculations, since all values are approximate anyway. Once we have a sufficient precision we stop; the sufficient precision depends on the implementation and the resolution of our character. This process is called *dichotomy* (from δίχα = in two pieces, and τέμνω = to cut), and is usually one of the first exercises in most programming languages. The code is shown below.

```
def solve_k_problem(suffix $, $$, $$$)(expr pen_width, first_try) =
pair z_zero; z_zero = z$;
numeric x_one, y_one, x_two;
y_one = y$$; x_one = first_try; x_two = x$$$;
numeric theta, n; n := 1;
forever:
        clearxy; z$ = z_zero; z$$ = (x_one, y_one);
        theta := angle(z$$ − z$); pos$$.1(pen_width, theta − 90);
        if y$ > y$$:
          z$$.1r
      else:
          z$$.1l
        fi = z$$;
    x_one := x_one
    if x$$.1r > x_two: −
    else: +
    fi abs(first_try − x$)/(2 ** n);
    exitif((abs(x$$.1r − x_two) < 0.1) or (n > 13)); n := n + 1;
endfor
pos$$$(pwidth, theta − 90); z$$$r = z$$.1r;
enddef;
```

This procedure expects that you feed it with: (a) the suffixes of points z_{1l}, z_{2l} and z_{2r}, (b) the width of the pen, (c) a hint on the first choice for x_{2l}. It usually works fine when your hint is simply the x-coordinate of z_{2r}. Theoretically it can go wrong if the x-projection of $z_{1l} - -z_{1r}$ is bigger than the first step of the iteration process. But this can hardly ever happen.

The iteration is stopped either (a) when the distance of z_{2r} to V is less than a tenth of a pixel, or (b) if $n = 14$, because the next step would produce a denominator $2^{15} = 32{,}768$, which is too big for (usual) METAFONT. Experience shows that with 8 steps one is usually done — again, all depends on the resolution.

Let's consider the second problem now. As the reader can see in fig. 3, only the position of point A differs. Nevertheless, this makes a big difference for METAFONT: in the previous problem, once we had chosen B' we could immediately calculate the position of C'. This is not the case here: all we know is that if D' is the middle of $B'C'$, then $AD' \perp B'C'$. So we need a different technique already to calculate the location of C' for each step of the iterating process. This will be done again by iteration.

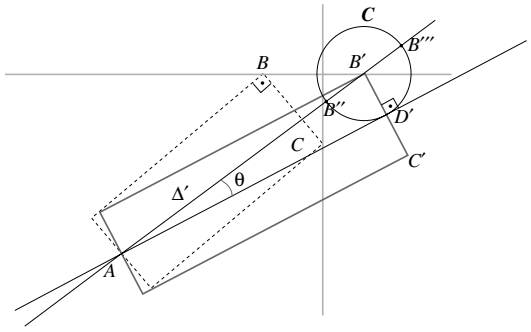


Figure 3: A closer look to the "*x problem*".

But first let's consider two geometrical facts which we are going to use.

Fact 1. *Let $\mathcal{C}$ be a circle, centered at O, and P a point outside the circle. For every line Δ, going through P and intersecting the circle at points X and X', the product of lengths $\overline{PX} \cdot \overline{PX'}$ is constant. In particular, if PT is a tangent to the circle going through P, then $\overline{PT}^2 = \overline{PX} \cdot \overline{PX'}$.*

As the reader can see on fig. 4, it follows from the previous fact that if Δ' is the line going through P and O, and intersecting the circle at Y and Y', then $\overline{PT} = \sqrt{\overline{PY} \cdot \overline{PY'}}$.

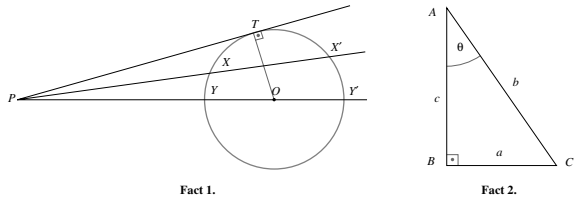


Figure 4: Two facts from elementary Euclidean geometry.

The second fact is even more trivial:

Fact 2. *Let ABC be an orthogonal triangle; the right angle shall be $\widehat{ABC}$; let's call $\widehat{CAB} = \theta$, and a, b, c the lengths of faces opposite to points A, B, C. Then $\theta = \arccos(\frac{c}{b})$.*

Let's return to our problem (see fig. 3). D' is on a circle $\mathcal{C}$ centered at B', of radius $\frac{1}{2}\overline{BC}$ (half the width of the stroke, since $\overline{B'D'} = \overline{D'C'}$). Also we know that $AD' \perp B'C' \Rightarrow AD' \perp B'C' \Rightarrow AD'$ is tangent to circle $\mathcal{C}$. Let's draw the line Δ, going through points A and B'. It will intersect $\mathcal{C}$ at points B'' and B'''. From **Fact 1** we know that $\overline{AB''} \cdot \overline{AB'''} = \overline{AD'}^2$. So we do not yet have D' itself, but the length of AD'.

Let's apply now **Fact 2** to the orthogonal triangle $AD'B'$. We obtain: $\theta = \arccos(\overline{AD'}/\overline{AB'})$. Once we have the angle and the length of AD', we have point D', and we are done for this step of the iterating process. The remainder of the solution is similar to that of the "*k problem*": we are moving B' around until C' is close enough to line V.

Let's try to implement this solution in METAFONT. **Fact 1** can be implemented easily: of course one should avoid multiplying two lengths (because of a possible overflow error), but there should be no problem if we take the square root of each length first (for purists: lengths are always positive!). So instead of $\overline{AD'}^2 = \overline{AB''} \cdot \overline{AB'''} =$ we will formulate the equation as $\overline{AD'} = \sqrt{\overline{AB''}} \cdot \sqrt{\overline{AB'''}}$.

Fact 2 is a little harder to implement. As a matter of fact, the reader may have noticed that although METAFONT provides exponential *and* logarithmic functions, there are no *inverse trigonometric* functions. What should be done? Unfortunately, METAFONT offers no complex calculus so that formulas such as $\cos(x) = \frac{1}{2}(e^{xi} + e^{-xi})$ could be applied; power series cannot be used either because our candidates for angles are not necessarily in a neighborhood of 0; using an external program to make this calculation would be highly unorthodox. Let's use dichotomy once again!

Here is the code for a *arccosd* procedure in METAFONT:

```
def arccosd(expr ttt) =
if ttt > 1:
    message("error: arccosd argument > 1!!??"); stop;
else: numeric a_;
    numeric test, nnn, prev; test := 45; nnn := 1;
    prev := cosd(test);
    forever:
        nnn := nnn + 1;
        if cosd(test) < ttt:
            test := test - (90/(2 ** nnn))
        else:
            test := test + (90/(2 ** nnn))
        fi;
        exitif((abs(test - prev) < 0.01) or (nnn > 14));
        prev := test;
    endfor
fia_ := test; enddef;
```

The above procedure requires as argument a number $ttt \in]0, 1[$. It stores the result in the numeric variable *a_*. The first try is always $\frac{\pi}{4}$. Since we want to solve a specific problem (the "*x problem*"), one must consider *arccosd* for only the first quadrant: solutions will always be in the range $]0, \frac{\pi}{2}[$. One can easily generalize the code to work in different ranges. In particular it would be nice to

modify the code to allow us getting results in the complex domain for $ttt \in]-\infty, 0[\cup]1, \infty[$ but the author can hardly see the utility of these for METAFONT...

Let us now have a look at the solution of the "*x problem*", as it is shown below.

A word of explanation concerning the pen position $$ is perhaps necessary. This is a quick way to obtain the intersection of line Δ' (fig. 3) and circle $\mathcal{C}$: the pen $$ lies on Δ' and points $$_r$ and $$_l$ are at the right distance from point $$. As we shall see in the following section this method yields results that are accurate enough for our purpose.

The same idea has been used to explicitly define point $$.1: by taking pen position $, the right edge of the pen is on point D' of the figure.

```
def solve_x_problem(suffix $, $$, $$$)(expr pen_width, first_try) =
pair z_zero;  z_zero = z$;
numeric x_one, y_one, x_two;
y_one = y$$;  x_one = first_try;  x_two = x$$$;
numeric theta, n, phi, tangent_length;  n := 1;
forever:
    clearxy;  z$ = z_zero;  z$$ = (x_one, y_one);
    theta := angle(z$$ - z$);
    pos$$(pen_width, theta);
    tangent_length := sqrt(length(z$$l - z$)) * sqrt(length(z$$r - z$));
    arccosd(tangent_length/ length(z$$ - z$));  phi := theta - a_;
    pos$(2tangent_length, phi);  z$$.1r = z$r + (z$r - z$$);  z$$.1l = z$$;
    x_one := x_one
        if x$$.1r > x_two: -
      else: +
      fi
      abs(first_try - x$)/(2 ** n);
    exitif((abs(x$$.1r - x_two) < 0.1) or (n > 13));  n := n + 1;
endfor
z$$$l = z$$.1l;  z$$$r = z$$.1r;  z$$$ = 0.5[z$$$l, z$$$r];
enddef;
```

The entries for this procedure are the same as for *solve_k_problem*. Again the problem is solved in the specific case of the right and upper part of letter 'X'; for the other possible cases, one should either change the code (straightforward but tedious), or use symmetry arguments.

1.3 A solution which doesn't work, and why

The author must confess that the first time he had to solve the "*x problem*" was during a METAFONT tutorial at the Royal Holloway College (UK) [this shows how badly the tutorial was prepared... *mea culpa*]. On the spot I could find no solution, yet on my way back across the Channel on the ship I found the code shown below. I am convinced that people taking the tunnel nowadays write better METAFONT code!

Let $\mathcal{C}$ be a circle, centered at B' and of radius $\frac{1}{2}\overline{BC}$, as in fig. 5. The intersection of Δ' and $\mathcal{C}$ gives us points B'' and B'''. From **Fact 1**, we obtain the length of AD'. Now, D' is at a known distance from A, and upon circle $\mathcal{C}$. Take a circle $\mathcal{C}'$, centered at point A and of radius $\overline{AD'}$. The (right) intersection of circles $\mathcal{C}$ and $\mathcal{C}'$ is the desired point D'.

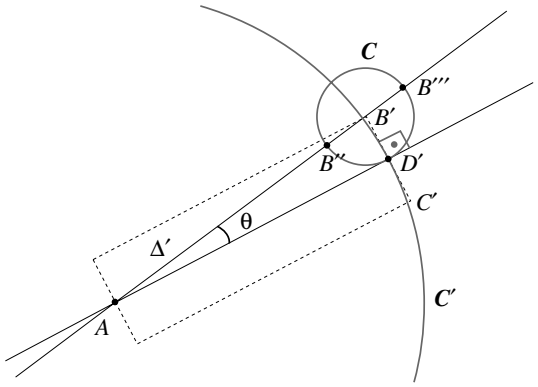


Figure 5: A solution of the "*x problem*" which doesn't work in METAFONT.

This method is mathematically correct — but if you try it out you will get extremely bad results.

The problem is that when we define a **path**, METAFONT does not consider it as an abstract curve, but as a set of pixels. When we ask for the intersection of two paths, we obtain the *pixel* which is the closest to the (theoretical) intersection of the paths. In the solution sketched above, the intersection of the two circles is taken as an abstract point, and its coordinates are used for calculations. The result is of course completely deformed.

2 Loosening Bézier curves

Bézier curves are quite beautiful, and METAFONT allows us to obtain them even out of only *partial* information: for example, one can ask for "a curve leaving point A following a vertical direction and arriving at point B following a horizontal direction". There is an infinite number of Bézier curves with exactly these features; METAFONT will choose one of them, out of hard-wired criteria. Most of the time, METAFONT's choice is exactly what you need; but it may also happen that you want to keep some other curve in the same set. There are two operators allowing us to do this:

1. `tension`, which allows us to get tense curves;
2. `controls`, by which we can explicitly determine the control points of our Bézier curve.

One can use `tension` quite intuitively: for a value of 1, the path remains unchanged; for higher values the path gets more and more tense. On the other hand, the operator `controls` gives us absolute control of the curve — but this is certainly not intuitive; maybe Leonardo da Vinci was smart enough to be able to guess the control point coordinates of Joconda's smile, but the rest of us would probably be unable to do it.

So it happened that the author often needed "loose" Bézier curves, and was unable to obtain them; unfortunately, `tension` doesn't work with values less than .75. (In fact, the METAFONTbook does not mention what the lower bound of the tension parameter is; however, repeated tries by the author have shown that the value .75 still works, while $.75 - \epsilon$ produces an error message.) With the following code, one can get arbitrarily loose Bézier curves:

```
def npush(expr p, coef) =
hide(pair firstpt, firstcpt, secondcpt, secondpt; firstcpt =
postcontrol 0 of p; secondcpt = precontrol 1 of p; firstpt = point 0 of p; secondpt
= point 1 of p; pair intersectpt, newfirstcpt, newsecondcpt;
intersectpt - firstpt = whatever * (firstcpt - firstpt);
intersectpt - secondpt = whatever * (secondcpt - secondpt);
newfirstcpt = coef[firstcpt, intersectpt]; newsecondcpt = coef[secondcpt, intersectpt]; )
firstpt .. controls newfirstcpt and newsecondcpt .. secondpt
enddef;
```

The macro *npush* takes two arguments: the path we want to loosen, and a numerical coefficient. For value 0 of this coefficient, the path remains unchanged. What happens when we increase this value? Let's consider the intersection point of the two Bézier tangents (the tangent at curve beginning and end). We know that control points always lie on these two tangents. For values between 0 and 1 of the coefficient, the control points travel between their original positions and the intersection point. For value 1 both control points are identified with the intersection point. For values higher than 1 they continue their travel outside of the Bézier triangle.

In Appendix A the reader can see the effects of the *npush* macro applied uniformly to all paths of a circle, with values of the coefficient going from -5 to 5.

As the reader has surely already noticed, this macro doesn't work when the tangents are parallel (because there is no Bézier triangle in that case). A second macro, with a slightly different approach covers all possible cases:

```
def mpush(expr p, lcoef, rcoef) =
hide(pair firstpt, firstcpt, secondcpt, secondpt; firstcpt =
postcontrol 0 of p; secondcpt = precontrol 1 of p; firstpt = point 0 of p; secondpt
= point 1 of p; pair newfirstcpt, newsecondcpt;
newfirstcpt - firstpt = lcoef * (firstcpt - firstpt);
newsecondcpt - secondpt = rcoef * (secondcpt - secondpt); )
firstpt .. controls newfirstcpt and newsecondcpt .. secondpt
enddef;
```

This macro has three arguments: the path which we will modify, and two numerical coefficients, corresponding to the transformation at curve beginning and at curve end. This time we multiply the distance of the first control point from curve beginning by the first coefficient, and that of the second control point from curve end by the second coefficient. Hence, in this case, value 1 for both coefficients will leave the path unchanged. For values higher than 1 the path will "swell", while for values tending to 0 it will become more and more tense.

These two macros may not be as reliable as primitive METAFONT operators, but they produce easily predictable results and are suitable for fine-tuning of character parts.

3 Getting text and numeric data out of a font

In his extremely interesting paper on communication between TEX and METAFONT [1], Alan Hoenig states that "*... METAFONT's file handling abilities are greatly crippled when compared to TEX. Other than font pixel files, font metric files, and log files, METAFONT cannot write files...*" To remedy that situation, we present `GFtoTXT`, a small utility for reading text (and any other information) out of GF files produced by METAFONT. As a matter of fact, METAFONT has a *special* command, just like TEX, but until now, no DVI driver was able to use it (as a possible use, one could very well imagine PostScript color instructions embedded in the GF files, taken over by the PK files and translated into real PostScript commands by the DVI driver).

`GFtoTXT` is written in Flex, a UNIX-originated lexical analyzer, under GNU copyleft. Flex allows the generation of highly reliable C code out of simple pattern matching, using regular expressions and states. The Flex code of `GFtoTXT` is very short; the reader can find it in Appendix B. To obtain an executable, (a) run this code through Flex, with the `-8` option — Flex will produce a C file called `lex.yy.c` (or `LEX_YY.C` on Messy-DOS systems); (b) compile it using your favourite C compiler. The Flex code as well as executables can be found on `ftp.ens.fr`, in `pub/tex/yannis/gftotxt`.

How does it work? There is one convention which must be followed: every string which we want to extract from the METAFONT run must start with the character `#`. This precaution is necessary because METAFONT itself sends several strings to the GF file by using internal *special* commands.

These will be ignored by `GFtoTXT`. Hence, to obtain the string "Hello world!" in our output file (let's call it `output.txt`), we will include the command

special(`"#Hello world!"`);

in the METAFONT code of our file (let's call it `input.mf`). `GFtoTXT` reads from the standard input flow and writes to the standard output flow, so we only need to redirect these; here is the necessary command line:

```
GFtoTXT < input.mf > output.txt
```

`GFtoTXT` allows three additional conventions in METAFONT strings: (a) `\n` will produce a carriage return in the output file, and (b) `\x` followed by a two-digit (lowercase) hexadecimal number between `00` and `ff` will produce the corresponding 8-bit character in the output file, (c) `\X` followed by a four-digit (lowercase) hexadecimal number between `0000` and `ffff` will produce the corresponding 16-bit character in the output file. Convention (a) is useful for "formatting" the output file, for example

special(`"#(CHAR C A\n (HEIGHT R "` & decimal h & `")\n (WIDTH R "` & decimal w & `")\n"`);

will produce

```
(CHAR C A
   (HEIGHT R 99.99976)
   (WIDTH R 129.99683)
```

Convention (b) is useful for inserting 8-bit characters (or characters in the range `"00`–`"1f`) into the output file. Finally, convention (c) can be useful to those of us who use UNICODE encoding: wouldn't it be nice to have METAFONT say to us ♡Σ' ἀγαπῶ♡? the necessary code would be

special(`"#\X2665\X03a3\X0027\X0020\X1f00\X03b3\X03b1\X03c0\X1ff6\X2665"`);

4 Bottom line

The different METAFONT techniques presented in this paper are certainly not programmed in the most elegant way; the author needed them for specific purposes, and stopped testing and refining whenever the specific problem was solved. It is *not* the goal of this paper to provide the reader with powerful new tools, but rather to stimulate him/her in creating his/her own, and to go beyond the `plain` and `cm` base macros. In all three examples, the basic idea was very simple: iterate calculations until a

sufficiently precise approximation is obtained, modify a path by manipulating the control points in the background, read the GF file by something else than GFtoPK or GFtoDVI. By writing down and sharing such ideas we can make out of METAFONT an even friendlier and more productive font design tool. To discuss METAFONT relative issues, but also font design in general (and why PostScript and TrueType are less efficient than METAfonts), join the METAFONT e-mail discussion list! The address of the list on the Internet is:

`metafont@ens.fr`

You can subscribe by sending the following subscription message to `listserv@ens.fr`:

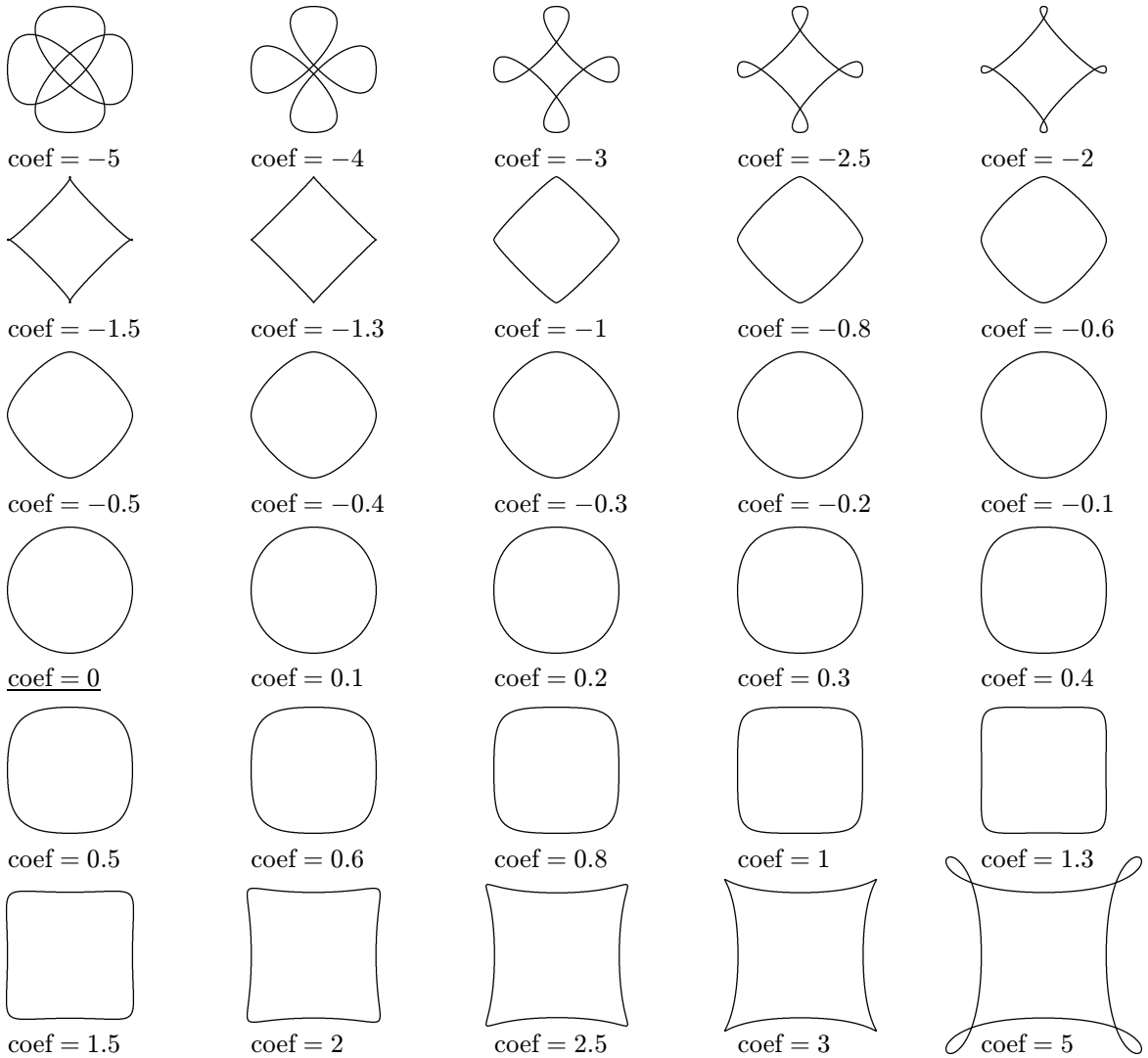
`SUBSCRIBE METAFONT` *Your name and institution*

References

[1] Hoenig, A. *When TEX and METAFONT talk: Typesetting on curved paths and other special effects*, pp. 549–553, *TUGboat* **12** (4), 1991

◇ Yannis Haralambous
187, rue Nationale, 59800 Lille, France.
Email: `haralambous@univ-lille1.fr`

A Transforming a circle through *npush*



B The Flex code of GFtoTXT

```
%{
#define HEXA(A,B) (yytext[(A)]>='a'? yytext[(A)]-'a'+10 : \
   yytext[(A)]-'0')*16 + (yytext[(B)]>='a'? yytext[(B)]-'a'+10 : yytext[(B)]-'0')
long int length_special = 0L; int we_need_it = 0, pre_length = 0;
%}

%x READ_SPECIAL
%x READ_PRE

%%

([\x00-\x3F\x45\x46\x4A-\xEE\xF4\xF8-\xFF])|([\x40\x47].)|([\x41\x48]..) ;

([\x42\x49]...)|(\x43.{24})|(\x44.....) ;

\xEF.#  { BEGIN READ_SPECIAL; length_special = yytext[1] - 1L; we_need_it=1; }

\xEF.   { BEGIN READ_SPECIAL; length_special = yytext[1]; we_need_it=0; }

\xF0..#   { BEGIN READ_SPECIAL; length_special = (yytext[1] * 256L) + yytext[2] - 1L;
      we_need_it=1;  }

\xF0..   { BEGIN READ_SPECIAL; length_special = (yytext[1] * 256L) + yytext[2];
      we_need_it=0;  }

\xF1...#   { BEGIN READ_SPECIAL; length_special = (((yytext[1] * 256L) +
      yytext[2]) * 256L) + yytext[3] - 1L; we_need_it=1;  }

\xF1...   { BEGIN READ_SPECIAL; length_special = (((yytext[1] * 256L) + yytext[2])
      * 256L) + yytext[3]; we_need_it=0;  }

\xF2....#   { BEGIN READ_SPECIAL;
      length_special = (((((yytext[1] * 256L) + yytext[2]) * 256L) + yytext[3]) * 256L)
      + yytext[4] - 1L; we_need_it=1;  }

\xF2....   { BEGIN READ_SPECIAL;
      length_special = (((((yytext[1] * 256L) + yytext[2]) * 256L) + yytext[3]) * 256L)
      + yytext[4]; we_need_it=0;  }

<READ_SPECIAL>.   { length_special--;  if (we_need_it==1) printf("%c",yytext[0]);
      if (length_special==0L) BEGIN 0; }

<READ_SPECIAL>\\x..   { length_special-=4;
      if (we_need_it==1) printf("%c",HEXA(2,3)); if (length_special==0L) BEGIN 0; }

<READ_SPECIAL>\\X....   { length_special-=6; if (we_need_it==1)
        printf("%c%c",HEXA(2,3),HEXA(4,5)); if (length_special==0L) BEGIN 0; }

<READ_SPECIAL>\\n   { length_special-=2;  if (we_need_it==1) printf("\n");
      if (length_special==0L) BEGIN 0; }

(\xF3....)|(\xF5.{17})|(\xF6.{10}) ;

\xF7.. {  pre_length = yytext[2]; BEGIN READ_PRE; }

<READ_PRE>.   {  pre_length--; if (pre_length == 0) BEGIN 0; }

.|\n ;

%%

main()
{
   yylex();
}
```

The Program a2ac — Font Handling on the PostScript Level

Petr Olšák

Abstract

In this article, the a2ac program written by the author is described. The program enables the use of PostScript fonts while typesetting texts in which accented letters are used. The font doesn't need to contain the complete alphabet of a given language; merely the presence of the accents themselves (and not all of the accented characters) is sufficient. The configuration files of the a2ac program are independent of the PostScript font encoding as well as of the typesetting system encoding. The program may be used to prepare a font for any typesetting system, but only TEX was tested. The a2ac package is available on CTAN including the source in C and the documentation. It can serve as an alternative to the fontinst program by Alan Jeffrey. The program was tested on the operating systems SUN OS, Linux, and DOS.

– – ∗ – –

1 Introduction

The program a2ac (Afm to Afm and add Composites) works with the afm file (Adobe Font Metrics) as its input and creates a new modified afm file as its output. The program reads a so-called "description file" during its work. The changes we want to make are defined in this file. The program adds the full description of new composites (accented letters) including kern pairs, etc., to the output afm file.

The afm file describes information about the letters using the symbolic name for each character (Aacute is the A with an acute accent, for example). Each character is described through one of two possible ways. Either the character name is bound to a definite encoding position and to a PostScript procedure for rendering the image of the character, or the character is described as a so-called "composite character". In this case the encoding position of the character is set to be −1 and the description of how to make the character from elements is stored in afm. The elements are usually characters described as mentioned previously and only their symbolic names (not their encoding positions) are used.

All desired new composite characters are defined in the description file. Since only the symbolic names are used, the description file is totally independent of the encoding of the PostScript font and of the encoding used by the typesetting system.

The program already has nonzero intelligence when it reads the description file. You can declare and use so-called "variables". You can write metric and composite information in the form of simple expressions. You can add new kern pairs using the information from already existing similar kern pairs (with exceptions being possible).

An example of the description file format is shown below.

```
NC Ecaron 2 ; PCC E 0 0 ; PAT caron 0 Carontop ;
NC Eacute 2 ; PCC E 0 0 ; PAT acute Acuteshift Acutetop ;
NK (Ecaron,Eacute) : E
NC ecaron 2 ; PCC e 0 0 ; PAC caron 0 0 ;
NC eacute 2 ; PCC e 0 0 ; PAC acute acuteshift vshift ;
NK (ecaron,eacute) : e
RK (P,T,V) ecaron 0
RK (P,T) eacute 0
```

In this example we define new letters Ě, É, ě and é. First, Ecaron and Eacute are defined as composite letters (NC lines), then new kerns are made for Ecaron and Eacute. The kerns are the same as the kerns for the letter E (NK line). Next, ecaron and eacute are defined as composite letters with the same kern information as the letter e (NC lines and NK line). Last, some exceptions for the kerns are specified. The pairs Pě, Tě and Vě will have no kern, in contrast to the pairs Pe, Te and Ve, where negative kerns are usually defined. The same exceptions are specified for the pairs Pé and Té.

For more information about the description file format see the a2ac-eng.doc file, which is included in the a2ac package.

Ligature information is not defined in the description file. We will show that it essentially doesn't matter. Ligatures are described in the input afm file at the end of lines with the C prefix as is shown in our following example:

```
C 102 ; WX 333 ; N f ; B 20 0 383 683 ; L i fi ; L l fl ;
```

The a2ac program just copies the ligature information to the output afm file. This is sufficient for generating the ligtables which are used by the typesetting systems. For example, the afm2tfm program reads this information and generates the proper data in the ligtable of the tfm format. New ligatures for TEX can be defined through the enc files. For instance, the LIGKERN information below is present in the file xl2.enc and defines new ligatures for the dash characters "--" and "---".

```
% LIGKERN hyphen hyphen =: endash ; endash hyphen =: emdash ;
```

2 Font preparation for TEX

After a2ac is used, preparation of the font for TEX continues on in the standard way (as for the English language). You can use the afm2tfm program, which reads the converted afm file and the encoding information file enc to match the internal TEX encoding. The result is a TEX virtual font which includes two kinds of information: the information about re-encoding from the internal TEX encoding to the raw PostScript font encoding (as was specified in the enc file) and the information about building the accented letters from the elements (as was specified in the "rebuilt" afm file).

Let us show an example of the PostScript font preparation for TEX for the Czech and Slovak languages. First, we run the a2ac program. The command line looks like this:

```
a2ac input.afm desc.tab output.afm
```

The first parameter is the name of the input afm file, the second one is the name of the description file and the third one is the name of the output afm file. The extensions (.afm and .tab) are obligatory — the program has no algorithm for adding the extensions automatically.

The command to run the a2ac program is usually part of a script or a batch file. For example, the UNIX script to convert the font from the afm format to the tfm format may look like this:

```
a2ac $1.afm cscorr.tab c$2.afm
afm2tfm c$2.afm -t xl2.enc -v c$2 r$2
vptovf c$2.vpl c$2.vf c$2.tfm
```

The name of this script is mkfnt, say. Then the usage is simple:

```
mkfnt Times-Roman ptmr
```

The file cscorr.tab is included in the a2ac package. The file includes definitions of composites for Czech and Slovak accented letters (i.e., the output font metric includes Czech and Slovak letters and the proper kerns for these letters). The input font has to contain the standard set of 26 Latin letters (lower- and uppercase) and all the accents necessary for the composite letters, which means the acute, caron, ring, quoteright, dieresis, and circumflex accents. If the Adobe font in StandardEncoding is used, then this requirement is satisfied. The output afm file then specifies all characters (plain or composite) of the Czech and Slovak alphabets and the new kern information.

The description file cscorr.tab was used without changes for all standard PostScript fonts (32 fonts installed in all RIPs) with very good results. Of course, if you want, you can rearrange the cscorr.tab file for your new font to obtain the best result.

If the above UNIX script is used, the TEX font suitable for Czech and Slovak texts is prepared in the $\mathcal{CS}$-font encoding, which is an extension of the Computer Modern font encoding (see Section "The TEX Encoding").

The output of the script is c*.tfm (for TEX input), r*.tfm and c*.vf (for dvips input). The only work we must do is place these files into the appropriate directories (or rearrange the script to make this work automatically) and add one line to the configuration file psfonts.map of the dvips driver. The line looks like this:

```
rfont PostScript-Font
```

If the PostScript font is not installed in the PostScript RIP of the output device, we have to store the pfb (pfa) format of the PostScript font in our computer. Then the line in psfont.map looks like this:

```
rfont PostScript-Font </path/to/file.pfb
```

or (for DOS):

```
rfont PostScript-Font <d:\path\to\file.pfb
```

The new font can be loaded into TEX using the \font command, as shown below. From that point onward, the font in question is available and text using the same encoding as the font can be conveniently typeset by TEX (and printed with dvips).

```
\font\newfont=cfont
{\newfont the eight-bit text in the same
          encoding as defined in xl2.enc
          can be used and the new font
          will be shown.}
```

3 The TEX encoding

The input text encoding for TEX doesn't need to be the same as the internal TEX encoding. The re-encoding can be performed by the `tcp` table of emTEX, by changes to the `xord`/`xchr` vectors in the `tex.ch` file (TEX's `WEB` source hacking) or by setting active characters. The `enc` file describes exactly the same encoding as the internal TEX encoding into which the hyphenation patterns have been read.

In the Czech Republic and the Slovak Republic, the so-called $\mathcal{CS}$-encoding (the encoding of the $\mathcal{CS}$-fonts) is used very often as the internal TEX encoding. The $\mathcal{CS}$-encoding is an extension of the Computer Modern text font encoding and follows the ISO-8859-2 layout for the Czech and Slovak letters. The $\mathcal{CS}$-fonts are included in the widely available TEX installation package with the name $\mathcal{CS}$TEX.

The advantages of the $\mathcal{CS}$-encoding (in comparison to the T1 encoding) from our point of view are the following:

- The fonts include the letters from the Czech and Slovak alphabets only. Font generation from the METAFONT sources is faster and the `pk` files are smaller. Exceptions (non-Czech/Slovak letters in names, for example) are accessible by `plain` TEX macros or by the `\accent` primitive.

- The $\mathcal{CS}$-font encoding is the conservative extension of the Computer Modern text fonts. Therefore, $\mathcal{CS}$-fonts can replace CM text fonts in the installation packages without problems and with backward compatibility. This feature saves space on the disk. No complicated macro redefinitions are necessary. For example, the `csplain` format, which is included in the $\mathcal{CS}$TEX package, gives the same results as the original `plain` TEX. There are only three differences: (1) The new letters with codes ≥ 128 are introduced by `\catcode`, `\lccode`, `\uccode`; (2) $\mathcal{CS}$-fonts are loaded instead of CM text fonts; and (3) the Czech and Slovak hyphenation patterns are read in addition to the US English ones.

- The layout of the accented letters in the $\mathcal{CS}$-font is the subset of the ISO-8859-2 standard. Therefore, it is possible to use an eight-bit input Czech/Slovak text without re-encoding in the UNIX environment. The power is in simplicity! The `tcp` table (emTEX) is used in the DOS installation because there are many different layouts for the Czech/Slovak alphabet and there is no fixed standard in DOS (in contrast to the UNIX-like systems).

- Kerning and other aspects of the fonts are language-dependent. The multi-language font (T1 encoded, for example) is nonsense from the typographer's point of view. In addition, we can't rely on the results of the font hacking from abroad.

The Computer Modern text font encoding (and its conservative extensions) is font-type dependent. For example, two different characters are encoded to the same position depending on `\rm` versus `\tt` font (the 'fi' ligature versus downarrow, for example) or on `\rm` versus `\it` font (the dollar versus the sterling). This feature implies some problems.

Two files, `xl2.enc` and `xt2.enc`, are included in the `a2ac` package. These files define extensions of the $\mathcal{CS}$-font encoding. The extensions include some characters which are not contained in the Czech and Slovak alphabets (i.e., in the $\mathcal{CS}$-font), but which are contained in the Adobe StandardEncoding material.

It is recommended to use `xl2.enc` for `\rm` and `\it`-like fonts and `xt2.enc` for `\tt`-like fonts. The dollar is in position 36 under any circumstances and position number 132 was given to the sterling. If you load the PostScript `\it`-like font then you need to know that the sterling is not accessible by the `plain` TEX `{\it\$}` but you need to use a macro with `\char132` code.

Position number 32 is not defined in the `xl2.enc` file because the cross accent for the Polish L and l is not included in the Adobe StandardEncoding. The Ł and ł themselves are in positions 163 and 179, respectively. If we need to use these characters in plainTEX with the PostScript font, the redefinition of the macros `\l` and `\L` is necessary.

Note that the three-letter ligatures are not included in the Adobe StandardEncoding. This doesn't cause any problems because the three-letter ligatures are not used in Czech and Slovak. If the font is available in the ExpertEncoding you can do some hacking at the `vpl` level to obtain these ligatures.

If you want to typeset math using the PostScript font, you need to edit the `vpl` file [1] to include the uppercase Greek characters taken usually from the Symbol font. Sorry, but it is not obvious how to typeset math with the PostScript font; some more hacking must be done — on the `plain` TEX macro level as well as on the `\fontdimen` font information. The pretty "look" of math is a result of a font designer's work and a lot of TEX-macro programming. "Its emphasis is on art and technology, as in the underlying Greek word" (see [2], Chapter 1.). Therefore, the pretty math is not accessible from the text PostScript font via any automatic process.

For more information about the $\mathcal{CS}$-font encoding and its history see [3] and appendix F in [4].

4 About the names of the accented letters

The names `dquoteright` and `tquoteright` should be used for the characters d' and t' in the description file `cscorr.tab`. However, we use the names `dcaron` and `tcaron` instead. The reason is that the uppercase alternatives of the characters d' and t' are Ď and Ť, i.e., `Dcaron` and `Tcaron`, respectively. The parameter `-V` (for `afm2tfm` to make the small caps variant of a font) does not work with the names `dquoteright` and `tquoteright`. We use `Lcaron` and `lcaron` instead of `Lquoteright` and `lquoteright` because the semantic for these accents is same as for `Dcaron`, `dcaron`, `Tcaron` and `tcaron`.

References

[1] Donald Knuth: Virtual fonts, more fun for grand wizards. *TUGboat* 11(1):13–23, April 1990.

[2] Donald Knuth: *The TEXbook*, vol. A of *Computers & Typesetting*. Addison-Wesley, Reading, MA, 1986.

[3] Petr Olšák: Úvaha o fontech v $\mathcal{CS}$TEXu [A reflection about fonts in $\mathcal{CS}$TEX], *TEXbulletin* (of $\mathcal{CS}$TUG) 3/93 (121–131).

[4] Petr Olšák: *Typografický sytém TEX* [Typesetting System TEX], $\mathcal{CS}$TUG 1995, 270 pages.

◇ Petr Olšák
Department of Mathematics
Czech Technical University in Prague
Czech Republic
Email: `olsak@math.feld.cvut.cz`

– – ∗ – – ∗ – – ∗ – – ∗ – – ∗ – – ∗ – – ∗ – – ∗ – – ∗ – –

Note added by production editor

In order to better understand how the above procedure works, I decided to download the program from `ftp.tex.ac.uk` in the directory `fonts/utilities/a2ac` below the CTAN root. I also got the `.afm` files for the PostScript fonts from the directory `fonts/psfonts/adobe`.

I now was all set to compile the `a2ac.c` program, and start to make the $\mathcal{CS}$-encoded files using the script `mkfnt`. It essentially runs the program `afm2tfm` that comes with the `dvips` distribution and creates `.tfm` and `.vf` files for TEX and `dvips` according to the desired encoding (see the Section "Font Preparation for TEX"). This was done for all "Standard" PostScript fonts. Figure 1 shows the font table for the font NewCenturySchlbk-Roman.

The righthand side of the table shows the characters specific to the Czech and Slovak languages. To obtain PostScript output from a `.dvi` file one more step has to be taken; namely, one has to tell the `dvips` program which file to read when a fontname is encountered. The easiest way of achieving this is to generate a new "config" file, since several can be specified with one `dvips` execution. In this case, I generated a one-line `config.cs` file containing

```
p +pscs.map
```

This states that the contents of the file `pscs.map` has to be concatenated with the contents of previously declared map files, in particular the default `psfonts.map`. The file `pscs.map` itself contains one line for each of the fonts considered: and starts like

```
rpagd   AvantGarde-Demi
rpagdo  AvantGarde-DemiOblique
rpagk   AvantGarde-Book
rpagko  AvantGarde-BookOblique
   ...
```

as explained in the text of the article.

As a further exercise, I decided it would be interesting to make this encoding available with LATEX, so I started by defining an encoding L2 in the file `L2enc.def` (See Chapter 7 of *The LATEX Companion*, by Goossens et al., Addison-Wesley, 1994 and the file `fntguide.tex` "LATEX 2_ε font selection", which is distributed electronically and comes with the standard LATEX 2_ε distribution.). This file contains merely:

```
\ProvidesFile{L2enc.def}%
              [1995/09/22 CS Encoding]
\DeclareFontEncoding{L2}{}{}
\DeclareFontSubstitution{L2}{ptm}{m}{n}
```

Next, one has to make *font definition* `.fd` files for each of the fonts one wants to use, e.g., for the Times-Roman family one would define a file `L2ptm.fd`, where `ptm` is a "short" name chosen in the framework of the font-selection scheme of Karl Berry (Version 2), which can represent all possible font names within the ISO-9660 (and DOS)-compatible eight characters limit for file names. Below the font definition file (`L2pag.fd`) for the L2 encoding of the AvantGarde font family is shown.

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
0		ı		0	@	P	‘	p	…	~	ì	˚	Ŕ	Ã	ŕ	
1			!	1	A	Q	a	q	†	¢	Ê	å	Á	Å	á	
2		`	”	2	B	R	b	r	‡	/	˘	¸	Â	Ň	â	ň
3		´	#	3	C	S	c	s	•	¥	Ł	ł	Û	Ó	ª	ó
4		ˇ	$	4	D	T	d	t	£	'	¤	´	Ä	Ô	ä	ô
5		¯	%	5	E	U	e	u	¶	·	Ľ	ľ	Ĺ	Ñ	Í	
6		¯	&	6	F	V	f	v	"	¸	*f*	ÿ	Ï	Ö	Õ	ö
7		°	’	7	G	W	g	w	<	º	§	ˇ	ï	Ÿ	õ	
8		¸	(	8	H	X	h	x	>	À	¨	â	Č	Ř	č	ř
9		ß	)	9	I	Y	i	y	\	ê	Š	š	É	Ů	é	ů
A		æ	*	:	J	Z	j	z	^	ò	è	Ò	È	Ú		ú
B		œ	+	;	K	[	k	–	_	û	Ť	ť	Ë	ù	ë	
C	fi	ø	,	¡	L	“	l	—	{	-	‹	Ì	Ě	Ü	ě	ü
D	fl	Æ	-	=	M	]	m	″	‰	¸	›	″	Í	Ý	í	ý
E		Œ	.	¿	N	ˆ	n	˜	|	«	Ž	ž	Î	ñ	î	„
F		Ø	/	?	O	˙	o	¨	}	»	ã	Ù	Ď	ß	ď	“

Figure 1: The layout of the $\mathcal{CS}$ encoded font

```
\ProvidesFile{L2pag.fd}[1995/09/22 L2 (CS) encoding for pag, MG]
\DeclareFontFamily{L2}{pag}{}
\DeclareFontShape{L2}{pag}{db}{n}{<-> cpagd}{}
\DeclareFontShape{L2}{pag}{db}{sl}{<-> cpagdo}{}
\DeclareFontShape{L2}{pag}{m}{n}{<-> cpagk}{}
\DeclareFontShape{L2}{pag}{m}{sl}{<-> cpagko}{}

\DeclareFontShape{L2}{pag}{b}{n}{<->ssub * pag/db/n}{}
\DeclareFontShape{L2}{pag}{bx}{n}{<->ssub * pag/b/n}{}
\DeclareFontShape{L2}{pag}{db}{sc}{<->ssub * pag/db/n}{}
      ....
```

The first line (`\ProvidesFile`) declares the name and version of the file to the LaTeX system (as it will appear in the `.log` and on the console). Then the family name `pag` for the L2 encoding is declared. This is followed by four declarations for the series and shapes for which actual external files are available, namely:

`pagd` AvantGarde-Demi
`pagdo` AvantGarde-DemiOblique
`pagk` AvantGarde-Book
`pagko` AvantGarde-BookOblique

The remaining declarations are substitution rules for combinations of font parameters which are not available as separate instances.

Finally, with such specifications for all used fonts, one can take a text encoded according to the given $\mathcal{CS}$-encoding, and run it through LaTeX. I have taken the test file distributed by Petr, and transformed it into LaTeX to test the above setup. Figure 2 shows the LaTeX source file. The command

```
dvips -E -Pcs L2test -o
```

yields the output shown in Figure 3, where the same Czech text has been typeset in Times-Roman, Times-Bold, Times-Roman-SmallCaps, AvantGarde-BookOblique, and finally AvantGarde-DemiOblique, showing clearly the advantages of this system.

◇ Michel Goossens
CERN, CN Division
CH-1211 Geneva 23
Switzerland
Email: goossens@cern.ch

```
\documentclass[a4paper]{article}
\usepackage[L2]{fontenc}
\renewcommand{\rmdefault}{ptm}   % Text font family is Times
\renewcommand{\sfdefault}{pag}   % San-serif font family is AvantGarde
\renewcommand{\ttdefault}{pcr}   % Teletype font family is Courier
\pagestyle{empty}
\begin{document}
                              ...  % text will be in Times-Roman
\bfseries                     ...  % text will be in Times-Bold
\mdseries\scshape             ...  % text will be in Times-Roman-SmallCaps
\sffamily\bfseries\itshape    ...  % text will be in AvantGarde-DemiOblique
        \mdseries\itshape     ...  % text will be in AvantGarde-BookOblique
\end{document}
```

Figure 2: LaTeX source of $\mathcal{CS}$ document

Tento text je napsán PostScriptovým fontem „*Times-Roman*“ za použití TEXovské metriky `cptmr`. Metrika fontu v české/slovenské variantě byla připravena konvertorem `a2ac` s následným použitím programu `afm2tfm`. Je potřeba si všímat méně obvyklých slov, jako je slovo šťastný (písmeno ť uprostřed slova), nebo třeba slovenské koľko. Věříme, že výsledný text půjde nejen přečíst, ale bude i typograficky pokud možno v pořádku. Také je třeba „sledovat chování“ našich „uvozovek“.

Á á Ä ä Č č Ď ď É é Ě ě Í í Ĺ Ĺ Ľ ľ Ň ň Ó ó Ô ô Ö ö Ř ř Ŕ ŕ Š š Ť ť Ú ú Ů ů Ü ü Ý ý Ž ž.

Tento text je napsán PostScriptovým fontem „*Times-Bold*“ za použití TEXovské metriky `cptmb`. Metrika fontu v české/slovenské variantě byla připravena konvertorem `a2ac` s následným použitím programu `afm2tfm`. Je potřeba si všímat méně obvyklých slov, jako je slovo šťastný (písmeno ť uprostřed slova), nebo třeba slovenské koľko. Věříme, že výsledný text půjde nejen přečíst, ale bude i typograficky pokud možno v pořádku. Také je třeba „sledovat chování“ našich „uvozovek“.

Á á Ä ä Č č Ď ď É é Ě ě Í í Ĺ Ĺ Ľ ľ Ň ň Ó ó Ô ô Ö ö Ř ř Ŕ ŕ Š š Ť ť Ú ú Ů ů Ü ü Ý ý Ž ž.

TENTO TEXT JE NAPSÁN POSTSCRIPTOVÝM FONTEM „*Times-Roman-SmallCaps*“ ZA POUŽITÍ TEXOVSKÉ METRIKY `cptmrc`. METRIKA FONTU V ČESKÉ/SLOVENSKÉ VARIANTĚ BYLA PŘIPRAVENA KONVERTOREM `a2ac` S NÁSLEDNÝM POUŽITÍM PROGRAMU `afm2tfm`. JE POTŘEBA SI VŠÍMAT MÉNĚ OBVYKLÝCH SLOV, JAKO JE SLOVO ŠŤASTNÝ (PÍSMENO Ť UPROSTŘED SLOVA), NEBO TŘEBA SLOVENSKÉ KOĽKO. VĚŘÍME, ŽE VÝSLEDNÝ TEXT PŮJDE NEJEN PŘEČÍST, ALE BUDE I TYPOGRAFICKY POKUD MOŽNO V POŘÁDKU. TAKÉ JE TŘEBA „SLEDOVAT CHOVÁNÍ“ NAŠICH „UVOZOVEK“.

Á á Ä ä Č č Ď ď É é Ě ě Í í Ĺ Ĺ Ľ ľ Ň ň Ó ó Ô ô Ö ö Ř ř Ŕ ŕ Š š Ť ť Ú ú Ů ů Ü ü Ý ý Ž ž.

Tento text je napsán PostScriptovým fontem „AvantGarde-BookOblique“ za použití TEXovské metriky `cpagdo`. Metrika fontu v české/slovenské variantě byla připravena konvertorem `a2ac` s následným použitím programu `afm2tfm`. Je potřeba si všímat méně obvyklých slov, jako je slovo šťastný (písmeno ť uprostřed slova), nebo třeba slovenské koľko. Věříme, že výsledný text půjde nejen přečíst, ale bude i typograficky pokud možno v pořádku. Také je třeba „sledovat chování“ našich „uvozovek“.

Á á Ä ä Č č Ď ď É é Ě ě Í í Ĺ Ĺ Ľ ľ Ň ň Ó ó Ô ô Ö ö Ř ř Ŕ ŕ Š š Ť ť Ú ú Ů ů Ü ü Ý ý Ž ž.

Tento text je napsán PostScriptovým fontem „AvantGarde-DemiOblique“ za použití TEXovské metriky `cpagko`. Metrika fontu v české/slovenské variantě byla připravena konvertorem `a2ac` s následným použitím programu `afm2tfm`. Je potřeba si všímat méně obvyklých slov, jako je slovo šťastný (písmeno ť uprostřed slova), nebo třeba slovenské koľko. Věříme, že výsledný text půjde nejen přečíst, ale bude i typograficky pokud možno v pořádku. Také je třeba „sledovat chování“ našich „uvozovek“.

Á á Ä ä Č č Ď ď É é Ě ě Í í Ĺ Ĺ Ľ ľ Ň ň Ó ó Ô ô Ö ö Ř ř Ŕ ŕ Š š Ť ť Ú ú Ů ů Ü ü Ý ý Ž ž.

Figure 3: Result of LaTeX run on document shown in Fig. 2

Problems of the conversion of METAFONT fonts to PostScript Type 1

Basil K. Malyshev

Abstract

The paper describes problems pertaining to the automatic conversion of METAFONT fonts into the PostScript Type 1 font format. Several methods of conversion are discussed. A short description of the *Paradissa Fonts Collection* is presented. It contains Computer Modern fonts (used in (LA)TEX) in ATM compatible PostScript Type 1 format. The use of the collection and the problems related to it are discussed.

– – * – –

1 Introduction

Intensive quantification of human activities often implies rapid modifications of many methods of data production, processing, and use. Thus it seems necessary to adapt the collected data to efficient modern processing techniques.

One of these problems is the conversion of fonts defined in METAFONT into the PostScript font format and reports on the automatic conversion of fonts described in the METAFONT format to the PostScript Type 3 font format have been appearing since 1987 (see Carr, 1987; Henderson, 1989; Hobby, 1989; Berry & Yanai, 1990). However, these results are little used due to the poor rasterization of PostScript Type 3 fonts at low and middle resolutions on widely used output devices. Furthermore, the fonts are incompatible with Adobe Type Manager (ATM) which essentially narrows their application area.

The PostScript Type 1 font format as published by Adobe (Adobe, 1990) allows the possibility of performing METAFONT → PostScript Type 1 conversion that promises high quality fonts with a wide application area (including ATM).

The methods for automating such conversion are discussed in this paper.

2 METAFONT and PostScript Type 1

When comparing METAFONT and PostScript Type 1, we note the following features.

METAFONT is a high level language for font description. To draw a letter, METAFONT describes the curve traveled by the center of the pen, and the shape of this pen is allowed to vary as the pen moves. Coordinates of all reference points can be defined by parameters and linear equations governing these parameters. The main advantage of this approach is that the same definition readily yields a family of infinitely many related fonts of type, each font being internally consistent. Thus, infinitely many typeface styles can be obtained from a single definition by changing only a few parameters. METAFONT does not dictate its own font parametrization technique, but provides a designer with all the necessary tools.

METAFONT does not satisfy up-to-date standards of rasterization, because it yields acceptable results only at resolutions higher than 300dpi. For example, 240dpi × 216dpi is already poor, and at about 100dpi, typical of many graphic displays, the results are quite unsatisfactory.

PostScript Type 1 is a low level tool for font description. It reduces the description of a character outline to lines and Bezier cubic curves specifying some additional information (declarative hints) for the rasterizer. It gives excellent results on printers and other graphical devices with PostScript interpreters, and on systems with ATM. Furthermore, the font structure of PostScript Type 1 allows an application to perform rasterization only on characters that are actually used in the document, and to do it much faster than by METAFONT.

Adobe Type 1 rasterizers can generate variant fonts at any affine transformation of an original font, but this is not sufficient to generate several typeface styles with different optical sizes, weights, widths, and so on from a single description. To solve this problem, Adobe Corporation has developed a *Multiple Master Extension of the Adobe Type 1 font format* which can be used to generate a wide variety of typeface styles from a single font program (Adobe, 1992). This font format contains several outline typefaces called *master designs* which describe one or more design axes. Design axes represent a dynamic range of one typographic parameter, such as the weight or width.

METAFONT, however, provides font parametrization tools which are more flexible and natural for a designer. Also, a designer can readily add new symbols to a completed font definition in such a way that they are automatically consistent with the old ones.

Thus, combining of flexibility of a font description in METAFONT with the high quality of rasterization and simple use of PostScript Type 1 fonts seems to be desirable. Such a link can be represented by an automatic METAFONT → PostScript Type 1 converter generating fonts of rather high quality.

3 Steps and methods of MF → Type 1 conversion

The task of converting METAFONT fonts into the PostScript Type 1 font format may be divided naturally into two rather independent steps:

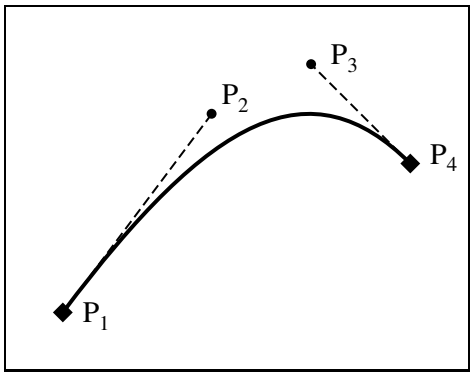


Figure 1: Bezier cubic curve defined by four points

- *extraction of character outlines* from the METAFONT font definitions;
- *generation of declarative hints*, which help the renderer to make best character rasterization.

The two steps can be performed by using different methods discussed below.

3.1 Extraction of character outlines

Each character in the PostScript Type 1 font is described by an outline specified by a set of lines and Bezier cubic curves. For a Bezier cubic curve four points are used: start point (P_1), endpoint (P_4), and two control points (P_2 and P_3), as shown in figure 1. The tangent vectors of endpoints are determined from the line segments P_1, P_2 and P_4, P_3. The algebraic equation for this curve is:

$$P(t) = (1-t)^3 P_1 + 3(1-t)^2 t P_2 + 3(1-t) t^2 P_3 + t^3 P_4$$

for $0 \le t \le 1$.

Note that the labelling used in figure 1 for endpoints and control points will then be conserved for all elements forming character outlines.

Character outlines of high quality PostScript Type 1 fonts should not only describe all glyphs with sufficient accuracy, but also satisfy some important rules formulated in *Adobe PostScript Type 1 font format* (Adobe, 1990).

A *Points at extremes.* An endpoint should be placed at most horizontal or vertical extremes. This means that most curves should not include more than 90 degrees of arc.

B *Tangent continuity.* Whenever one path element makes a smooth transition to the next element, the endpoint joining the two elements and the Bezier control point associated with that endpoint (for a curve) or the other endpoint (for a line) should all be collinear.

C *Consistency.* All character features (stem width, height, spacing, shapes) that are intended to be the same should be exactly the same.

D *Conciseness.* Character outline definitions must be as concise as possible, without breaking the other rules.

There are two main approaches to the extraction of character outlines

- generation of character raster images by METAFONT with a subsequent recovering of the outlines by *tracing the pixels* and approximating the resulting outline by lines and Bezier cubic curves (see section 3.1.1);
- *direct extraction of outlines* from the METAFONT program with subsequent removal of overlapping elements and geometric optimization of resulted outline (see section 3.1.2).

3.1.1 Tracing the pixels of the bitmaps

At first, we need to generate bitmapped fonts by METAFONT. In PostScript Type 1 font format the endpoints and control points are defined on a 1000×1000 grid. To avoid rounding errors after outline scaling, magnification should be chosen so that font design size is rasterized into 1000 pixels. Therefore, the resolution should be chosen as

$$\langle \text{resolution} \rangle \frac{dots}{\text{inch}} = \frac{72.27\text{pt/inch} \times 1000\, dots}{\langle \text{design size} \rangle \text{pt}}.$$

Thus, the best resolution for the design size of 10pt is 7227dpi. This resolution does not require additional scaling of the resulting outline, and the METAFONT program performs correct coordinate rounding.

It is easy to make METAFONT itself to compute the required magnification by adding

```
pixels_per_inch:=4000 + 3227;
pixels_per_inch:=pixels_per_inch*4pt#;
tmp:=designsize/2.5;
pixels_per_inch:=pixels_per_inch/tmp;
```

to `mode_def` macro definition in the `local.mf` file.

Many programs will reconstruct a raster image outlines by tracing the pixels. However, font generation requires that the outlines satisfy the rules discussed in section 3.1. Below we consider several sets of results, and focus on whether these rules are obeyed or violated.

The software developed by Neil Raine generates the outlines from the bitmaps by tracing the pixels. Graham Toal generated Computer Modern fonts at 3000dpi resolution and produced PostScript Type 3 fonts by using Raine's program. These fonts are available in CTAN in `/tex-archive/fonts/cm/ps-type3` directory. The examples of these outlines are

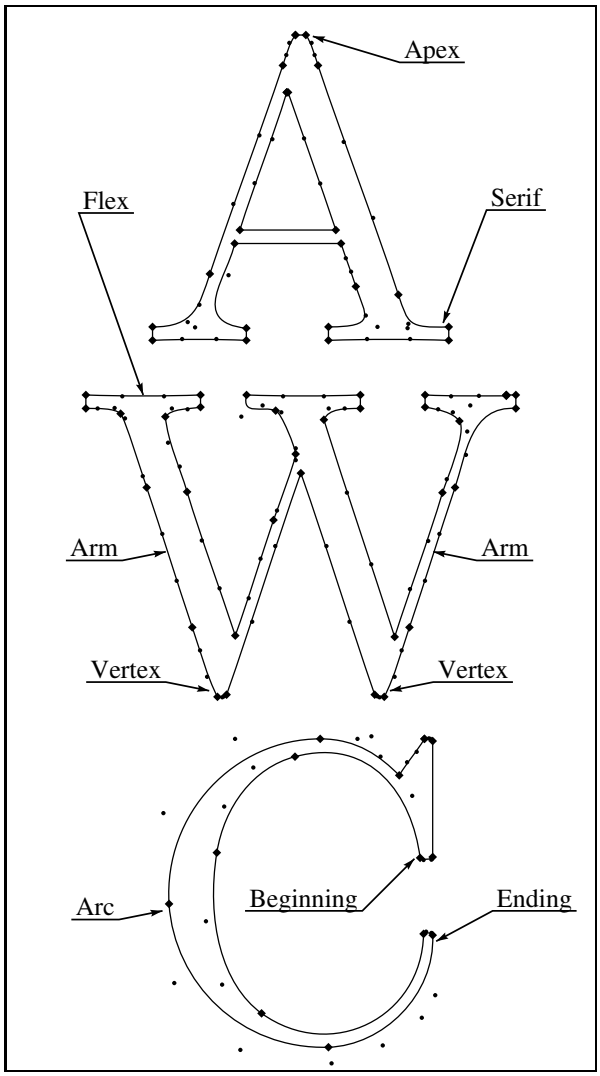


Figure 2: Neil Raine and Graham Toal cmr10 outlines

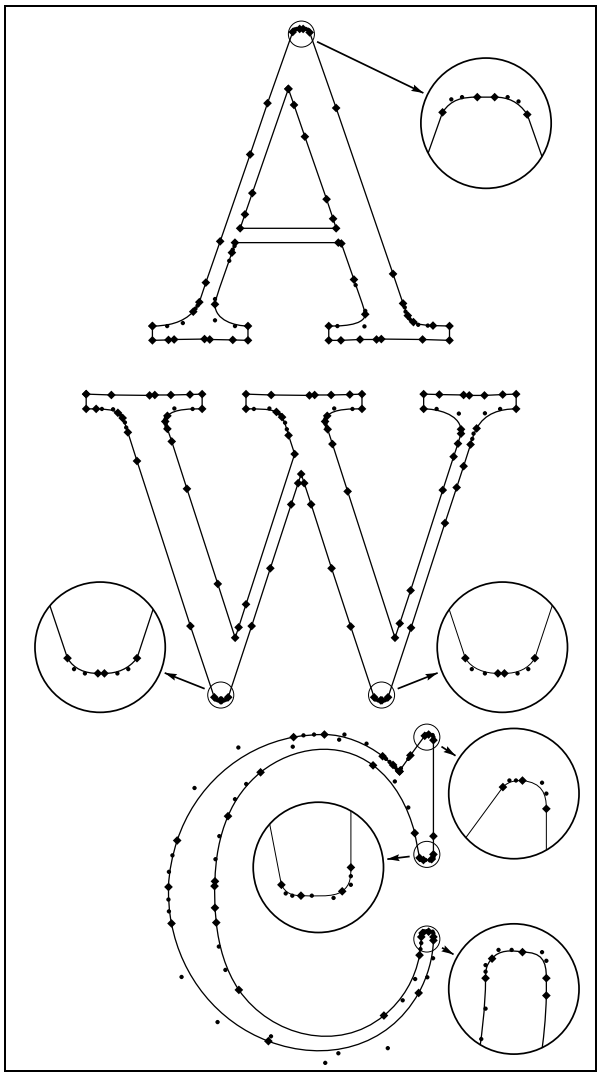


Figure 3: cmr10 outlines produced by `LIMN` from the GNU FONT UTILITIES

shown in figure 2. It is easy to see that these outlines suffer from some defects:

- The violation of rule **3.1.A** in the letter 'C'. This defect is not occasional, but is common to most curved stems.
- Asymmetrical *vertices* in the letter 'W' violate rule **3.1.C**.
- Bad transitions from *arms* to *serifs* in the letters 'A' and 'W'.
- A middle *serif* in the letter 'W' is unsatisfactory.

Karl Berry and Kathryn Hargreaves developed the GNU FONT UTILITIES, and announced them in TEXhax (Volume 92, Issue 8 and 17). These utilities contain a program `LIMN` which takes the bitmap fonts and generates outlines by tracing the pixels. The results of their operation are shown in figure 3.

The outlines presented in this figure also have some visible defects:

- The violation of rule **3.1.A** in the letter 'C'. Similar to the above example, this defect is regularly repeated.
- Each *flex* is split into many short line segments, violating rule **3.1.D**.
- Multiple consecutive collinear line segments violate rule **3.1.D**.

These outlines, when compared with Toal's, offer some evident advantages, namely the apex of the letter 'A' is symmetrical and the vertices of the letter 'W' are symmetrical and match each other.

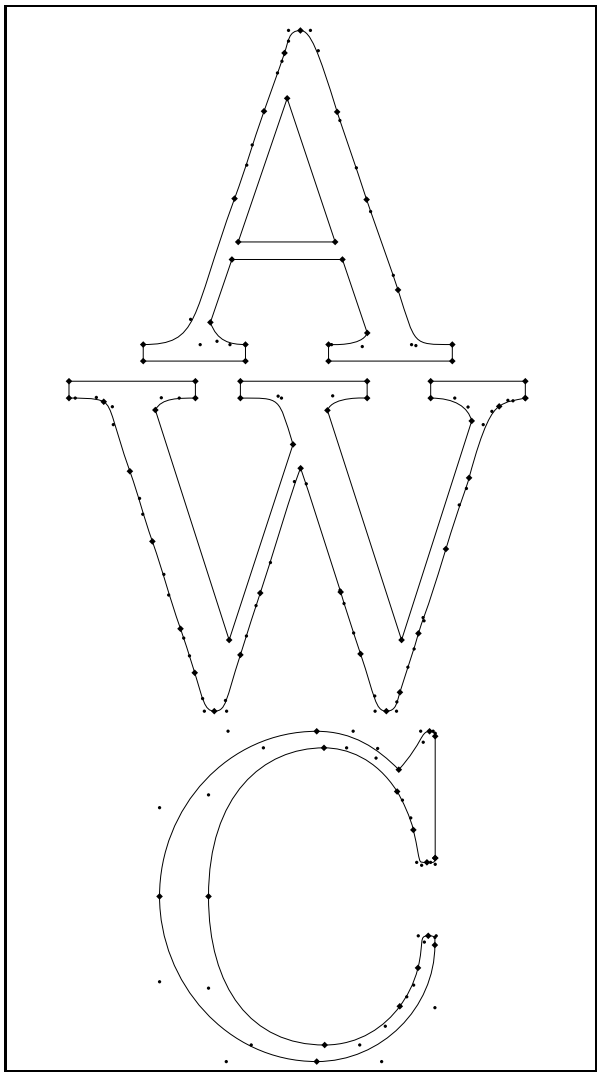


Figure 4: cmr10 outlines produced by `FontoGrapher`

To make more comparisons, bitmap fonts (at 3613dpi resolution) were traced by `FontoGrapher` (version 3.5.1). The resolution 3613dpi = 7227dpi/2 has been chosen because the tracing procedure built in `FontoGrapher` can break at too high a resolution. The result of this experiment is shown in figure 4. In spite of the fact that rule **3.1.A** is satisfied here (having been violated in the above examples), the outlines are not free from some defects:

- The *apex* in the letter 'A' is asymmetrical (violation of rule **3.1.C**).
- The *vertices* in the letter 'W' are asymmetrical and different (violation of rule **3.1.C**).
- *Arm – serif* transitions in the letters 'A' and 'W' are not satisfactory.

Evidently, it is difficult to obey rule **3.1.C** when directly tracing raster images. On the other hand, rule **3.1.A** seems to be easy to satisfy, and the violation of this important rule in the first two examples could probably be attributed to lack of authors' attention to it.

3.1.2 Extraction of outlines from METAFONT

Each program considered in section 3.1.1 exhibits its own defects of outline generation. Besides, there are defects common to all programs, such as poor discovery of *flex* features or bad *serif–arm* transitions.

The information critical for good appearance of characters is evidently lost when tracing an outline on a bitmap. Therefore, the extraction of outlines from METAFONT definitions without raster representation of fonts seems to be more fruitful.

The first attempt at extracting the character envelopes from METAFONT was undertaken by Leslie Carr (1987). Carr's programs take as input the METAFONT log file which contains a description of all the paths that METAFONT traces out in drawing a character. But using this method one should take into account the METAFONT pen shape.

Later Daniel M. Berry and Shimon Yanai (1990) have developed a more successful program, `mf2ps`, that finds the internally generated METAFONT envelopes, used as the boundaries of the inked region, and uses these envelopes as the PostScript outlines. In both these attempts, PostScript Type 3 fonts have been generated. The outlines generated by the `mf2ps` program are shown in figure 5.

To present such envelopes for PostScript Type 3 font the `mf2ps` program reorders cycled subpaths and chooses black and white filling for each of them. This method, although suitable for Type 3 fonts, fails for Type 1.

To produce outlines suitable for Type 1 fonts, all envelope overlapping should be removed. The result of this operation is shown in figure 6. Note that the resulting outlines contain too many consecutive lines and curves split into many pieces. To obtain outlines free from such defects, I have made some geometrical optimization. The result is shown in figure 7.

Now the outlines look significantly better than those in figures 2, 3 and 4. All the rules from section 3.1 have been satisfied. The fragmentation of the inner side of arc in the letter 'C' occurs because of the high (probably too high) accuracy in the approximation of the original shape.

There are some slight peculiarities related to a different encoding of similar shapes. In the letter

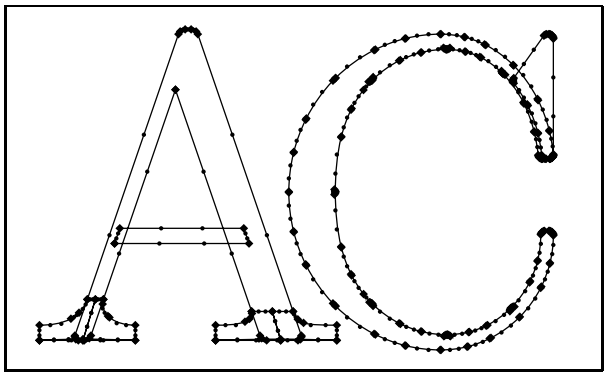


Figure 5: METAFONT internally generated envelopes

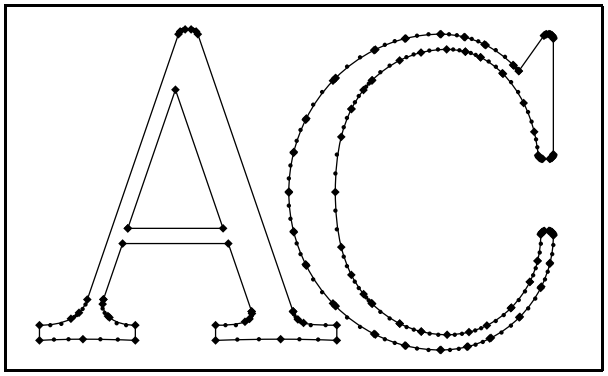


Figure 6: METAFONT envelopes after removing overlapping

'Φ', for example (figure 8), the outline representation is somewhat asymmetrical. The outline itself has a slight asymmetry, but in the process of rasterization its asymmetry of encoding may be exaggerated.

Nevertheless, this method is a step forward in improving the character outlines obtained from the METAFONT font definitions.

3.2 Generation of declarative hints

One of the main problems arising in font rasterization on a discrete grid is the conservation of the important geometrical properties of outlines. Identical parts of the letters differently located on the grid may take different shapes in a discrete representation.

In the PostScript Type 1 font format this problem is solved by using *declarative hints* which indicate where a horizontal or vertical stem occurs in certain coordinates. Those parts of the outlines which appear inside of the so-called stem hints will be rendered by special techniques.

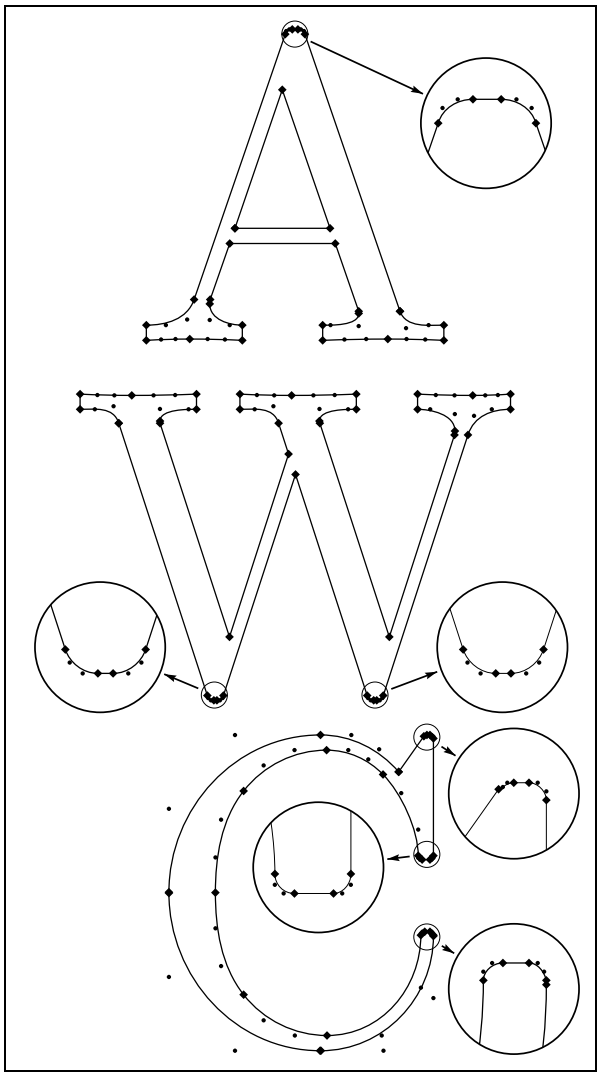


Figure 7: METAFONT envelopes after geometrical optimization

Declarative hints for the outlines obtained at the previous stage can be generated by font editors like `FontMonger` or `FontoGrapher` as follows:

- styling the outlines in the form of ATM compatible font format;
- loading the outlines to a font editor;
- saving the completed font on a disk.

As a result, the saved font contains the declarative hints.

The hints generated by `FontMonger` (version 1.0.4) are shown in figure 9. Comparing these outlines with the original ones from figure 3, it is easy to see that `FontMonger` has changed the outlines so that rule **3.1.A** is satisfied. However, stem hints for the serifs in the letters 'A' and 'W' are missing.

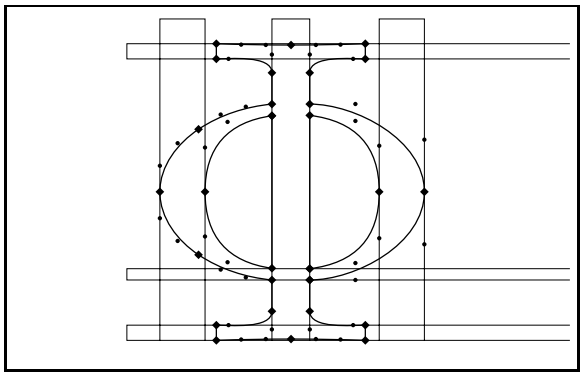


Figure 8: Asymmetrical coding of the like symmetrical form in the letter Φ

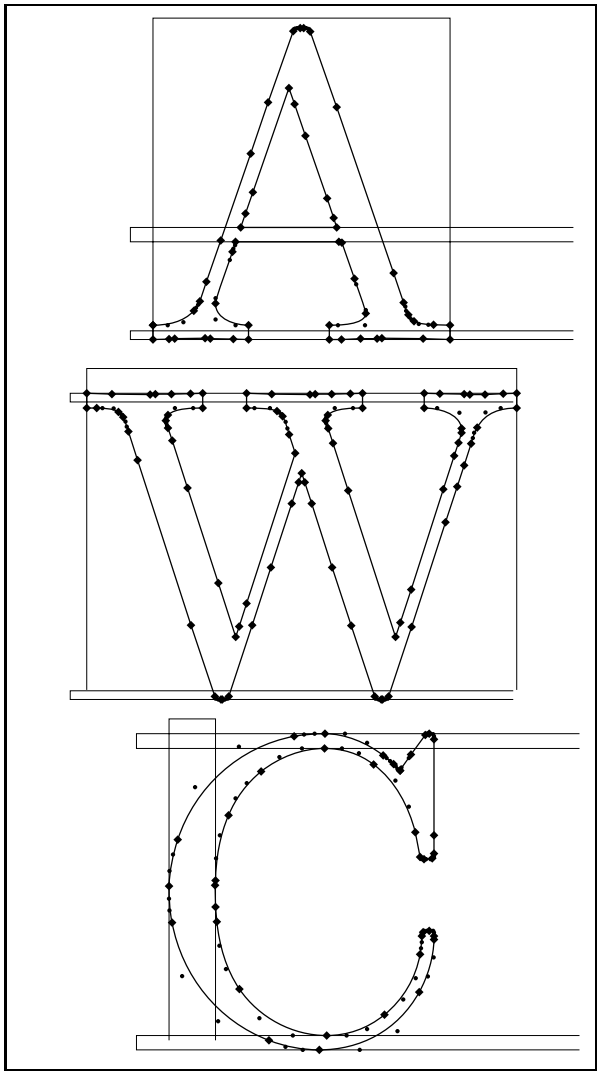


Figure 9: `LIMN` generated outlines hinted by `FontMonger`

The declarative hints generated by `FontoGrapher` for the same outlines are shown in figure 10. In this figure the outlines are unchanged, while the hints for the curved stems in the letter 'C' are missing. That is, `FontoGrapher` does not even try to correct the violation of rule **3.1.A**, and this violation has a pernicious effect on its operation. However, unlike the latter case, all hints for the serifs in the letters 'A' and 'W' are found correctly now.

If the outlines are not too accurate (such as those generated by the `LIMN` program), then the operation of both `FontMonger` and `FontoGrapher` is not good enough. However, if a font processed by `FontMonger` is subsequently processed by `FontoGrapher`, the obtained results can be significantly better (figure 11).

In the case of more accurate outlines directly extracted from METAFONT (figure 7), this approach is quite suitable, but `FontMonger` can hardly find the serifs (figure 12), while the results obtained by `FontoGrapher` are quite acceptable (figure 13).

Thus, one can see that `FontoGrapher` performs hinting somewhat better than `FontMonger`. Still `FontMonger` also has an advantage useful for mass conversion of fonts because it has a batch conversion utility.

The *Paradissa Font Collection* has been created using a homegrown algorithm for generating character hints. This algorithm has been developed especially for processing the outlines generated by the `LIMN` program. The result of its operation is shown in figure 14 where it can be seen that almost all the hints have been found, but the outlines have not been corrected as in the case of running `FontMonger` (figure 9).

In the case of outlines directly extracted from METAFONT (figure 7), our homegrown algorithm gives results (figure 15) competitive with those obtained by `FontoGrapher` (figure 13).

4 Paradissa Font Collection

The *Paradissa Font Collection* has been developed using the outlines generated by `LIMN` and a specially developed outline filter and hinting algorithm. This font collection is available from CTAN in the

`/tex-archive/fonts/cm/ps-type1/paradissa`

directory. The examples of the hinted outlines contained in this collection have already been presented in figure 14.

The Paradissa Font Collection contains:

- Computer Modern, designed by D. Knuth;
- Euler by H. Zapf;
- CM Cyrillic by N. Glonty & A. Samarin;

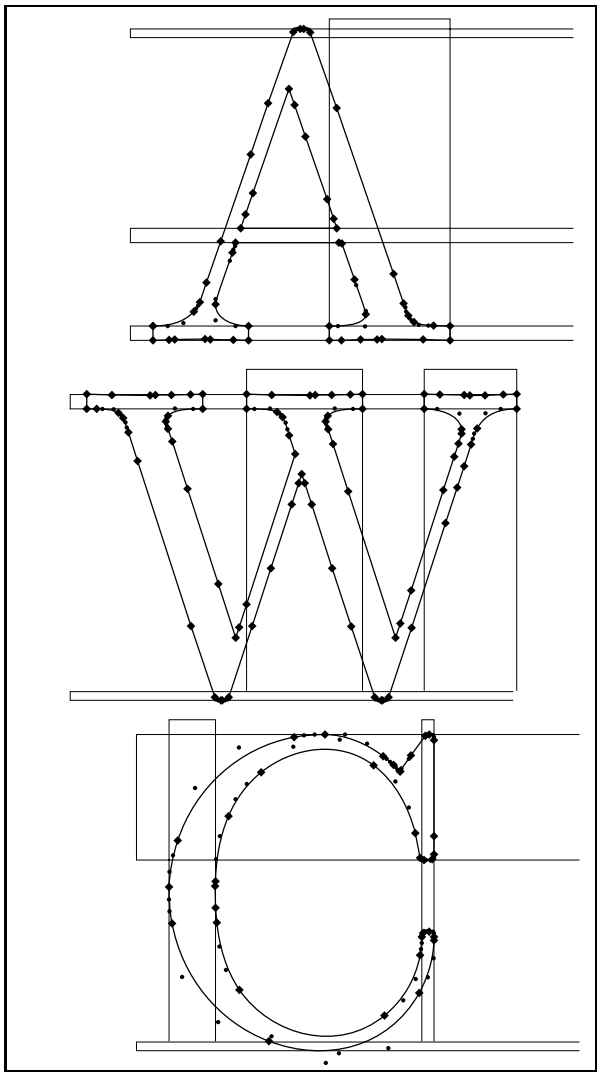


Figure 10: LIMN generated outlines hinted by FontoGrapher

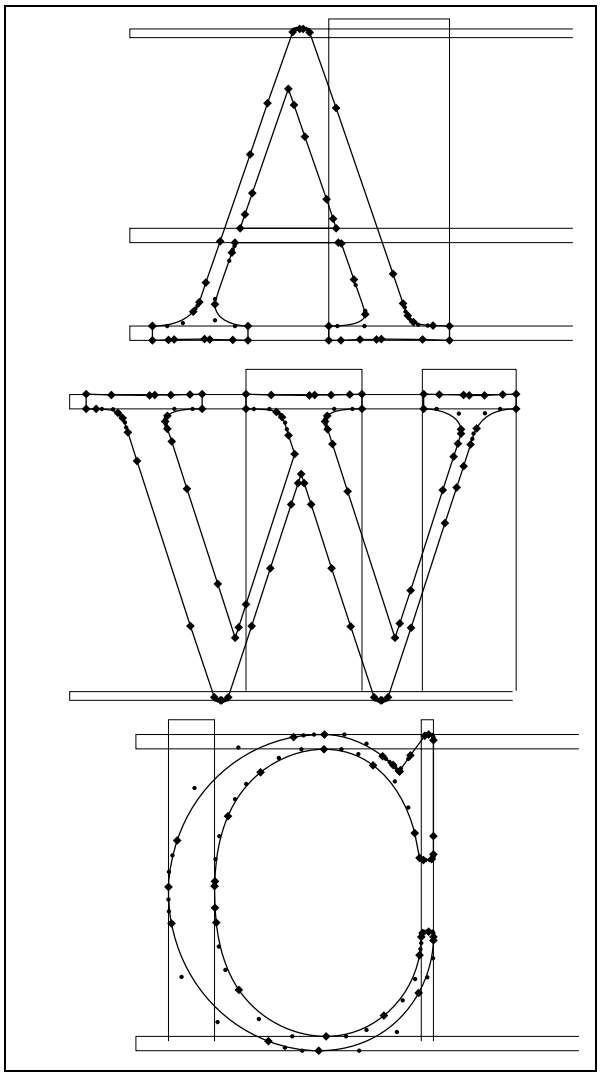


Figure 11: LIMN generated outlines hinted first by FontMonger and then by FontoGrapher

- Special LaTeX fonts.

Altogether it contains 165 fonts in ATM compatible PostScript Type 1 format with AFM and PFM files. This set of fonts may be used for printing most (La)TeX documents. It is used by the Russian-English LaTeX version developed and supported by the ProTeX group at IHEP.

This collection can be used for

- printing documents on a PostScript printer by using, for example, Rokicki's DVIPS driver. It should be noted that the typesetting of even a simple LaTeX document may require a lot of printer memory to download fonts. This problem is solved, for instance, by the commercial program DVIPSONE which uses a special technique for *partial font downloading* to conserve the printer's memory.
- printing documents on a large collection of matrix printers by using DVIPS and ghostscript.
- drawing slides on vector plotters by using the PostScript plot.ps program which is supplied with the collection. For drawing documents on HPGL plotters, the ps2hpgl utility can be used. It is available on ftp.mathworks.com host in the /pub/contrib/tools directory.
- displaying documents under MS Windows with ATM by using the commercial DVIWindo program. We also expect that the capability of using Type 1 fonts will be added to Hippocrates Sendoukas' DVIWIN program.

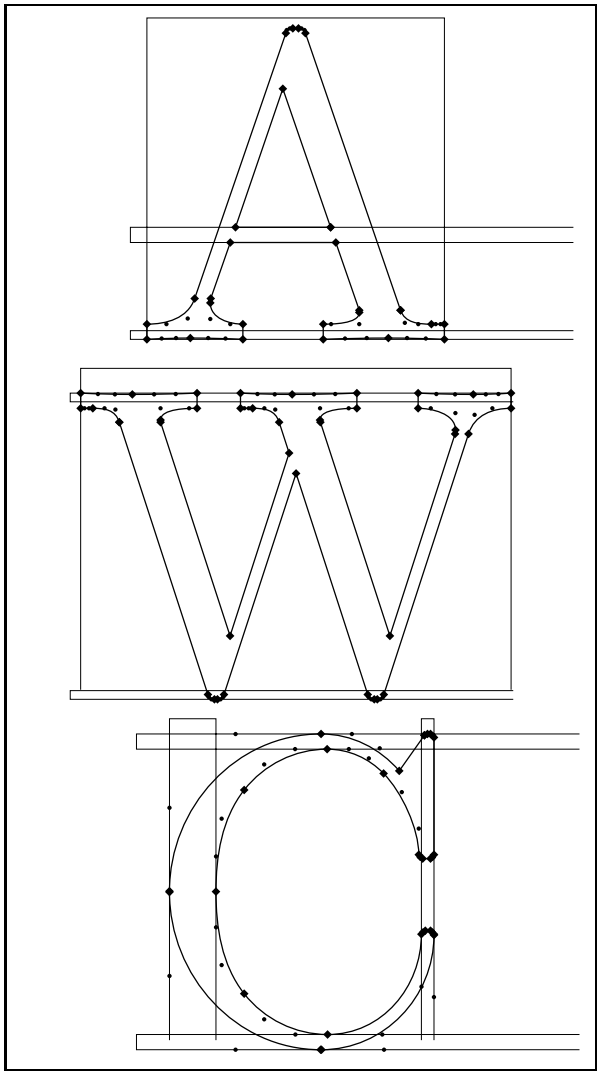


Figure 12: BKM outlines hinted by `FontMonger`

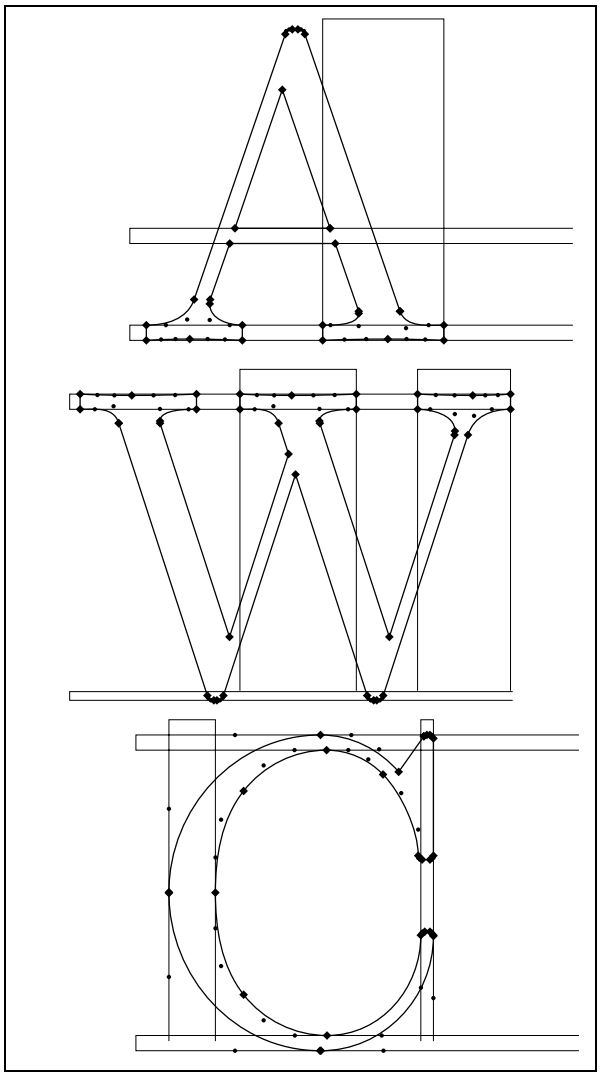


Figure 13: BKM outlines hinted by `FontoGrapher`

References

Adobe Systems Inc. *PostScript Language Reference Manual*. Addison-Wesley, 1985.

Adobe Systems Inc. *Adobe Type 1 Font Format*. Addison-Wesley, August 1990, Version 1.1.

Adobe Systems Inc. *Adobe Type 1 Font Format: Multiple Master Extensions*. Adobe Developer Support, 14 February 1992.

Berry, Daniel, and Shimon Yanai. "Environment for Translating METAFONT to PostScript." *TUGboat* **11** (4), p. 525–541, 1990.

Carr, Leslie. "Of Metafont and PostScript." *TEXniques* **5**, p. 141–152, August, 1987.

Henderson, Doug. "Outline fonts with METAFONT." *TUGboat* **10** (1), p. 36–38, 1989.

Hobby, John D. "A METAFONT–like System with PostScript Output." *TUGboat* **10** (4), p. 505–512, 1989.

Knuth, Donald E. *The METAFONTbook*. Reading, Mass.: Addison-Wesley, 1986.

Knuth, Donald E. *METAFONT: The Program*. Reading, Mass.: Addison-Wesley, 1987.

◇ Basil K. Malyshev
Institute for High Energy Physics,
IHEP, OMVT, Moscow Region,
RU-142284 Protvino, Russia
Email: `malyshev@mx.ihep.su`

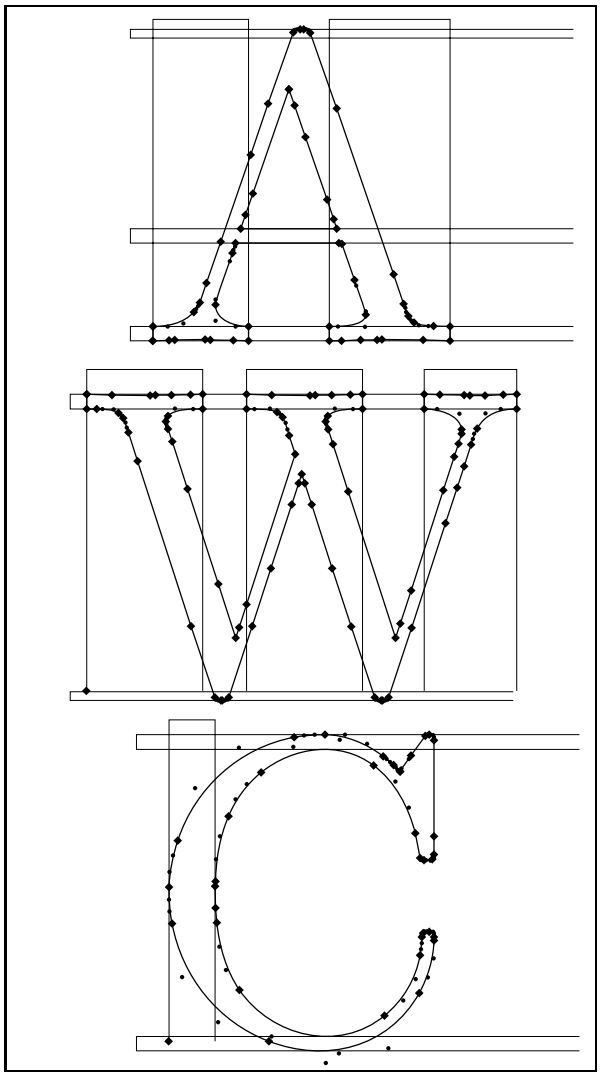


Figure 14: LIMN generated outlines hinted by homegrown hinter

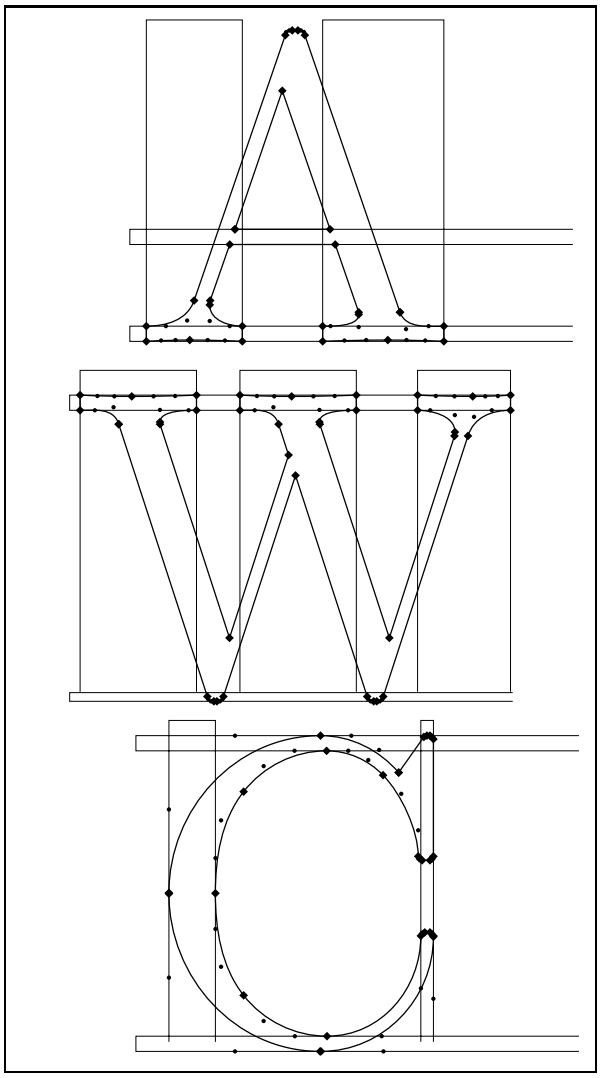


Figure 15: BKM outlines hinted by homegrown hinter

Partial Font Embedding Utilities for PostScript Type 1 Fonts

Basil K. Malyshev and Michel Goossens

Abstract

This article describes the `fload` utility, that reads a PostScript file and selectively embeds only those characters from Type 1 PostScript fonts that are actually used inside the PostScript file. This can result in substantial savings in the size of PostScript files, especially for larger font sets (like the 256-character DC family) or when only a few characters of each font are used inside the PostScript file. Also memory requirements for PostScript interpreters dealing with these files can thus be reduced.

1 Introduction

Documents often use a lot of fonts and thus require printers with a sufficient amount of memory, especially if the fonts are embedded as Type 1 sources inside the output PostScript file. However, in most cases only a limited number of characters of each font is actually used, and PostScript files with embedded Type 1 fonts can thus be substantially shortened if only those characters are included.

The tool `fload` uses the `ghostscript` program (Aladdin Software) to examine the input PostScript file and extract the necessary information; then it embeds the needed characters in the output PostScript file.

Partial font embedding is implemented in three steps, as follows:

1. Creation of a PostScript file using any publishing system. *No* fonts should be embedded in this file; the `fload` procedure assumes that the Type 1 images of the fonts are available somewhere on the system.
2. The PostScript file is processed by the program `ghostscript` where some character loading commands, such as `show`, `awidthshow`, etc., have been redefined to construct the list of all characters used for each font. When the complete input file is read these font usage statistics are written into an intermediate file (with extension `fstat`).
3. The latter file is then read by the `subfont` program which embeds the needed characters for each font into the output file.

This approach can be applied to PostScript output generated by a variety of applications—it has been tested with `dvips`, FrameMaker, and MS Windows PostScript driver output. Moreover it handles fonts inside embedded pictures correctly. Its only drawback is that `fload` needs some time to scan the input file to prepare the list of characters used and then to embed the (partial) fonts into the output file.

2 Setting up the `fload` system

2.1 The distribution set

The `fload` distribution includes the following:

1. A command file (a C-shell script `fload` for Unix, `fload.bat` for MS-DOS, and `fload.com` for VMS) that first invokes `ghostscript` with the `psfstat.ps` procedure and then executes the `subfont` program to embed the needed characters (see section 3.2).
2. A PostScript procedure `psfstat.ps` which redefines several important PostScript commands and collects information about the characters used in each font.
3. The `subfont` program (written in C) which extracts only the required characters from any ATM compatible Adobe Type 1 font and creates a new smaller font (see appendix A).

2.2 Installation

To make installation of the programs easy for Unix systems a C-shell script `INSTALL` is included. Before running the script, one must specify where certain programs and files reside, by editing the file `install.dirs` (see below).

```
# File: install.dirs
# System dependent executable files
# e.g. the subfont program
setenv FBIN /usr/local/bin
# System independent scripts
# e.g. the fload C-shell script
setenv FSHR /usr/local/bin
# Library files
# e.g. psfstat.ps, Fontmap, SubFont.map, *.FS
setenv FLIB /usr/local/lib/fload
# man pages (optional)
setenv FMAN /usr/local/man/man1
# font path
setenv FPATH myfonts:/usr/local/lib/texmf/fonts
# End of file 'install.dirs'
```

The installation then proceeds as follows:

1. Compilation of the `subfont` program.
2. The `subfont` program shares a single fontmap file (called `Fontmap.t1` in the `$FLIB` directory) with the `ghostscript` program. It is read by the `psfstat.ps` script in addition to the standard `Fontmap` file required by `ghostscript` to indicate where the font sources are located on the system; the format of this file is identical to a `ghostscript` map file and includes a list

of the font names and the associated file names where they reside, i.e., it is of the form:

```
/Type1_fontname (filename_on_system) ;
```

A Unix script `mkmap` (`mkmap.bat` on MS-DOS) is provided to create a map of the PostScript Type 1 files available in the specified directory tree, as shown in the following example:

```
[1] mkmap /usr/local/lib/texmf/fonts
/usr/local/lib/texmf/fonts
Total of 1 files are processed.
font /cmb10/ in file /./cm/type1/cmb10.pfb/
Total of 1 files are processed.
font /cmr10/ in file /./cm/type1/cmr10.pfb/
Total of 1 files are processed.
...
```

This procedure generates the file `Fontmap.t1`, that starts with (compare with the lines above):

```
/cmb10 (./cm/type1/cmb10.pfb) ;
/cmr10 (./cm/type1/cmr10.pfb) ;
...
```

3. All files from the `lib` distribution subdirectory are copied into the `$FLIB` library directory. The `INSTALL` script as distributed defines `$FLIB` as the directory `/usr/local/lib/fload`. In principle, `$FLIB` could also be defined as `/usr/local/lib/ghostscript` (the directory used by `ghostscript`). However, to minimize possible future name clashes between file names for `fload` and `ghostscript`, it is advisable to keep these directories distinct.
4. The names of fonts that are *not* to be included in the output PostScript file are the ones specified via the `-p` option of `fload`. By default these are the thirteen "standard" fonts, as specified in the file `Standard.FS` that comes with the distribution. Another distributed file, `35.FS` contains the names of the 35 "generally available" PostScript fonts present in most of the present-day PostScript printers. At the end of installation, users can add their own supplementary `FS` files specifying the names of fonts that are resident in their PostScript printer, and that thus need not be embedded in the generated output PostScript file. For instance, in Russia Cyrillic fonts could always be resident, or Kanji fonts in Japan or China. To save considerable amounts of disk space these fonts will not generally need to be embedded in the PostScript files, and their names must then be declared in a file with extension `FS`, and specified with the `-p` option of the `fload` command (see section 3.2). Of course, when these files have to be exported to sites where these fonts are not available on the system, they will have to be embedded in the PostScript files. `FS` files reside in the `$FLIB` directory.

3 Using the `fload` program

This section shows how to set up `dvips` to generate output usable with `fload` and then introduces the syntax of the `fload` command itself.

3.1 Preparing PostScript output with `dvips`

To use `fload` with the `dvips` program (Rokicki) one must use a map file for `dvips` that specifies that PostScript fonts are *not* to be embedded into the PostScript output file. In particular, the map file must include only font names, and *no* file names, i.e., entries should be of the form (file `cmfontsr.map`):

```
cmr10 cmr10
```

and not (file `cmfonts.map`):

```
cmr10 cmr10 <cmr10.pfb
```

since the latter specifies that the font should be input from the file `cmr10.pfb`. Note, however, that `subfont` must know where it can find the fonts it needs, so that, in this case, `cmr10` must be declared in the `ghostscript` font map (see step 2 in section 2.2).

3.2 Syntax of the `fload` command

`fload` is a C-shell script that parses (via the procedure `psfstat.ps` and the `ghostscript` interpreter) a legal PostScript file, determines which characters in which fonts are required but *not preloaded* in the file,[1] embeds ("loads") the required characters from the fonts in question (using the `subfont` program), and writes them together with the input PostScript source file to the output file.

```
fload [-p fset] <PSf-in> [<PSf-out>]
```

`-p fset`

This option selects the font set that is resident in the printer, and from which the characters will be taken. The standard configuration comes with two font sets, namely `Standard` (the default) and `35`. The user can easily add more font sets (see section 2.2 step 4).

The list of all font sets installed at a given installation can be obtained by calling `fload` without parameters, as shown below.

```
[1] fload
Usage: (of the fload de Basil)
fload [-p fontSet] <ps-file> [<out-ps-file>]
```

[1] Partial font downloading with `fload` only works when the font files in question *are not* already embedded in the input file `<PSf-in>`.

```
  where following fontSets are available
35
Standard
```

`<PSf-in>`
: Input PostScript file that has to be scanned to define the characters to be embedded.

`<PSf-out>`
: Output PostScript file containing the Type 1 definitions of the characters in the fonts referenced in the document. If `<PSf-out>` is not given the file is written to standard output.

If one has an invalid PostScript file, or one of the required fonts is not available, then a file with extension `flog` can be examined; it contains the transcript of parsing the source by `ghostscript`.

The `fload` procedure has been tested with PostScript output generated by `dvips` (see Appendices B to F, and figures 1 and 2), FrameBuilder (on Unix), and the MS Windows PostScript driver. Also, several kinds of fonts have been embedded, such as ATM compatible Type 1 fonts, Multiple master fonts,[2] and fonts with shareable sets of CharStrings. Note, however, that every new font must be tested to ensure that partial font embedding functions correctly.

4 Examples

As explained above to partially load only characters actually used in their document, (LA)TEX users should proceed in two stages:

1. run `dvips` on the `dvi` file using a map-file that does *not* embed the fonts inside in the PostScript output file (see section 3.1);
2. run `fload` on that PostScript file to embed the characters used inside the documents, so that the file becomes portable (see section 3.2).

Below we give several explicit examples.

4.1 CM fonts only

We start (Appendix B) by treating a simple one-page document (the French document described on page 333 of *The LaTeX Companion* (Goossens et al.), and reproduced also in figure 1). After running LaTeX we generate the PostScript file with `dvips` and include the header file `draft.ps` (via the `-h` command option) to put the word "DRAFT" across the page in the "funny" `cmff10` font (see figure 1 on page 77 for more details). In fact, we run `dvips` three times to show different ways of embedding the fonts. We start by using the default setup that includes the `pk` version of the fonts `[1]`.[3] Next `[2]`, we embed the complete font inside the output file using the `-P` option (using the configuration file `config.CM` that includes `cmfonts.map`), and finally `[3]`, we run with the "resident" version of the config file (`config.CMr` and its associated map file `cmfontsr.map`), that does not embed fonts (see section 3.1 for more about these `dvips` map files).

The listfile (`ls`) command `[4]` shows that the Type 1 fonts take up more than 90% of the file (that is normal for a one-page document, and not a representative number for "normal" situations). On the other hand, the `pk` versions of the fonts (at the default resolution of 300 dpi) take up about half the size of the file (31kb out of 59kb).

Now we are ready to extract the characters explicitly referenced from the Type 1 font files and embed them by running the `fload` utility `[5]`, and another listfile `[6]` tells us that the Type 1 fonts now take up about 3/4 of the file.

Finally we have a look inside the intermediate file `babelfrer.fstat`, that the `fload` program writes (with the help of `ghostscript`) to see which characters in which fonts are used in the input PostScript file `babelfrer.ps`. One observes not only the presence of the "normal" LaTeX fonts, but also the `cmff10` font that was included via the header file `draft.ps`, and for which the five letters (D, F, A, R, and T) are all present in the list.

4.2 Using DC fonts

Appendix C deals with a LaTeX file similar to the one in appendix B, but we now use 8-bit characters in the input, the T1-encoding, and DC-fonts. The numbering of the commands corresponds to that in Appendix B. It is seen that, at the given resolution, the `pk` file and "reduced" Type 1 version are only marginally larger than with the CM fonts, while the full font embedded version is almost twice as large as its CM equivalent, which is as expected since a DC font contains about twice as many characters as its CM equivalent. The embedded characters are shown in the file `babelfre8r.fstat`. When one compares the entry for a CM font like `cmr10` in the file `babelfrer` in Appendix B with its DC equivalent `dcr10` in the file `babelfre8r.fstat` in Appendix C, one notes the occurrence of the precomposed Eacute, agrave, eacute, etc., characters, that are absent from CM fonts, where they are constructed using the `\accent` primitive.

[2] Multiple master fonts are treated correctly only with `ghostscript` version 3.12 or higher.

[3] The numbers between square brackets in this section refer to the command numbers in the appendices.

4.3 DC fonts and some maths

The example in appendix D (it corresponds to figure 11.9 in *The LaTeX Companion*, German edition) also uses 8-bit input and DC-fonts, but contains also a lot of mathematics (see figure 2). In this case it is especially interesting to note the huge difference `[3]` between the three files without (`mathexar.ps`), with partial (`mathexadcr.ps`), and with complete font embedding (`mathexa.ps`). The contents of the file `mathexar.fstat` shows the actual characters that were embedded.

4.4 Real-life examples

To have a better idea of the usefulness of the `fload` procedure in real life, we have generated PostScript output files under the various conditions discussed earlier, but generating for a Linotype typesetter (at 1270 dpi), which corresponds more to a professional environment where the advantages of the approach proposed in this article are more evident.

Appendix E shows how a version of the present article's dvi-file (`fload.dvi`) was first generated with the `pk` fonts `[1]`, embedding the full Type 1 fonts `[2]`, and embedding no fonts `[3]`. The latter file was treated by `fload` `[4]`, and the sizes of the various files can be seen by comparing the result of the listfile command `[5]`. It is seen that in this case the sizes of the `pk` and Type 1 version are more or less comparable, while the file treated with the `fload` procedure is more than 30% smaller than both of these. This means that partial font-embedding is especially important size-wise for medium sized documents, using a few characters in many fonts, and at professional resolutions. Note, however, that the (partial font loading) Type 1 solution is resolution-independent, so that the relative advantages should not be limited to the size-parameter only.

As a final test, we had a look at the $\mathcal{AMS}$-distribution (American Mathematical Society), and treated two of their documents (`amsldoc.tex` and `testmath.tex`, see Appendix F). The output generated by the `dvips` command `[1]` shows the 29 CM fonts that are included. The listfont command `[2]` shows the difference between including complete Type 1 fonts (file `amsldoc.ps`) and embedding only the Type 1 code for the characters actually used (file `amsldocr.ps`). It is seen that the reduction is of the order of two in three. The next line `[3]` is even more informative, since it corresponds to a real article in the field of mathematics. The PostScript code for the article, without the fonts (file `testmathnone.ps`) is about 181 kbytes. When we add complete Type 1 fonts (file `testmath.ps`), we end up with almost 1.6 Mbytes, while with the `fload` program, this is reduced to just under half a Mbyte (file `testmathr.ps`). This has to be compared to the situation using pk-fonts at a resolution of 1270 dpi (file `testmath1270pk.ps`), as normally used for $\mathcal{AMS}$ journals, where one needs just over 2 Mbytes.

5 Conclusion

From all of these examples, especially from the real-life ones in section 4.4, it is evident that substantial gains in file size can be obtained by using the `fload` system. This is in particular the case for high resolution typesetting, where `pk` font bitmaps become rather large, while the size of the (reduced) Type 1 fonts remain identical (apart from small resolution-dependent differences in positioning introduced by the `dvips` program). This reduction in size is also important if one wants to export the PostScript file for use in other applications, like Adobe Acrobat, where Type 1 fonts are gracefully handled, while support for Type 3 fonts is not very good. It would be helpful if this kind of partial font embedding could become somehow more closely integrated with the `dvi` driver itself.

References

Aladdin Software. *GhostScript version 3.33 interpreter/previewer*. Electronically distributed document, 1994.

American Mathematical Society. *$\mathcal{AMS}$-LaTeX Version 1.2 User's Guide* and *Sample Paper for the `amsmath` Package*. Electronically distributed documents, 1995.

Michel Goossens, Frank Mittelbach and Alexander Samarin. *The LaTeX Companion*. Addison Wesley, 1994.

Tomas Rokicki. *Dvips: A DVI-to-PostScript Translator*. Electronically distributed document, 1994.

Appendices

A Syntax of the `subfont` program

The `subfont` program parses an Adobe Type 1 font program (in `pfa` (hexadecimal) or `pfb` (binary) format), extracts the specified characters, and writes a new font containing only those characters. When working with a single font it outputs a font in a `pfa` or `pfb` `atm` compatible format. When working in multi-font mode (the default or chosen with the `-f` option) the output is written in a form that can only be used by a complete PostScript interpreter (not by `atm`). Moreover, in multi-font mode, fonts with the same encoding share one encoding vector,

the `eexec` PostScript operator is not used in the output stream, and several other optimizations are performed.

The syntax of the `subfont` program is:

```
subfont command-list <file>
```

If instead of a file name `<file>`, one uses a hyphen (-), then the input Type 1 font is read on standard input. This is useful for using `subfont` in command pipes.

`-v` Run in *verbose* mode, i.e., output information during the parsing stage.

`-o<file>`
Write the output to file `<file>`. By default, the output is written to `stdout`, from where it can be easily redirected.

`-a<file>`
Similar to `-o`, but the output file is opened in append mode. This can be useful for adding a prolog.

`-b<file>`
Write the output file in binary mode. This only works in single-font mode, where it writes a `pfb` file. If used in multi-font mode, the program will stop when one tries to load the second font.

`-c<file>`
Copy the file from input to output (as-is). Can be useful on VMS.

`-n/../../`
This option specifies the list of characters to be included. If it is absent, all characters are included. The result will be to convert a `pfb` to a `pfa` file, or the reverse (see an example below).

`-f <file>`
Reads the list of the required characters and fonts from the file specified and processes each font in turn. This option activates multi-font mode. When fonts have an identical encoding they will share their encoding vectors. The `eexec` operator will not be used in the output stream and other optimizations will be performed. Every line of this file must contain a specification of the type:

```
font-name/char-1/char-2/.../char.../
```

Examples can be seen in appendices B, C, and D, in the listings of the files with extension `fstat`.

`-m<map-file>`
Specifies the name of the file containing the mapping between the font names, as used in the file processed by the `-f` option, and the files containing the fonts in question. The format of a map file is described in section 2.2, point 2.

`-I<path>`
The font search path used for searching the font files (used with the `-f` option).

`-u<file>`
The name of the file that will contain the list of fonts that are undefined. This file will be created only in case unknown fonts exist (used with `-f` option).

`-e#` Touch encoding flag. In multi-font mode identical font encoding vectors are loaded only once. To do this, every encoding vector is loaded out of the font body and must have a unique name and every font must only refer to an already loaded encoding vector.

To turn off this feature one should use the option `-e-1`. When the program prepares several files without the `-f` option but using the `-n` option and `<file>`, the encoding vector will be loaded as-is.

To force removing identical encodings one can say `-e1`.

`-x#` `eexec` applying flag. In multi-font mode fonts will be output without the `eexec` instruction. This feature makes output files smaller.

To suppress this feature in multi-font mode one can use the option `-x2`. When the program prepares several files without the `-f` option but using the `-n` option and `<file>`, then `eexec` will be applied. In this case, to remove `eexec` one can use `-x0`.

`-z <file1> ...`
Read every file in the list `<file1> ...` and for each line with the pattern:

```
/FontName /<font-name> def
```

output a line like:

```
/<fontname> (<file-name>) ;
```

This facility is useful for making font map files and is actually used by the `mkmap` utility described in section 2.2 (point 2).

As an example, one can create a font containing only the characters "T", "E", and "X" from the Courier font, by specifying:

```
subfont -b TEX.pfb -n/T/E/X/ cour.pfb
```

where the compressed binary `pfb` format is used.

One can select the alphanumeric output format by typing:

```
subfont -o TEX.pfa -n/T/E/X/ cour.pfb
```

If one wants a representation where the character outline code is not encoded with `eexec`, one can use the `-x0` option:

```
subfont -x0 -o TEX.pfa -n/T/E/X/ cour.pfb
```

B Example of using `fload` with the CM fonts

```
[1] dvips -hdraft.ps babelfre -o babelfrecm.ps
This is dvipsk 5.58e Copyright 1986, 1994 Radical Eye Software
' TeX output 1995.01.29:1928' -> babelfrecm.ps
<draft.ps><tex.pro><special.pro>. [1<colorcir.eps><tac2dim.eps>]
[2] dvips -hdraft.ps -PCM babelfre -o babelfre.ps
This is dvipsk 5.58e Copyright 1986, 1994 Radical Eye Software
' TeX output 1995.01.29:1928' -> babelfre.ps
<draft.ps><tex.pro><cmr12.pfb><cmbx12.pfb><cmbx10.pfb><cmr10.pfb><cmmi10.pfb>
<cmtt10.pfb><cmr7.pfb><cmti10.pfb><texps.pro><special.pro>. [1<colorcir.eps>
<tac2dim.eps>]
[3] dvips -hdraft.ps -PCMr babelfre -o babelfrer.ps
This is dvipsk 5.58e Copyright 1986, 1994 Radical Eye Software
' TeX output 1995.01.29:1928' -> babelfrer.ps
<draft.ps><tex.pro><texps.pro><special.pro>. [1<colorcir.eps><tac2dim.eps>]
[4] ls -l babelfre*.ps
-rw-r--r--   1 goossens et         408759 Jan 29 19:37 babelfre.ps
-rw-r--r--   1 goossens et          58818 Jan 29 19:37 babelfrecm.ps
-rw-r--r--   1 goossens et          28209 Jan 29 19:35 babelfrer.ps
[5] fload babelfrer.ps > babelfrerr.ps
Process PS file babelfrer.ps.
Write font using statistic to babelfrer.fstat
Process font 'cmti10' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmti10.pfb'
Process font 'cmff10' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmff10.pfb'
Process font 'cmr10' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmr10.pfb'
Process font 'cmbx10' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmbx10.pfb'
Process font 'cmr12' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmr12.pfb'
Process font 'cmtt10' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmtt10.pfb'
Process font 'cmmi10' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmmi10.pfb'
Process font 'cmr7' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmr7.pfb'
Process font 'cmbx12' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmbx12.pfb'
Total of 10 files are processed.
[6] ls -l babelfrerr.ps
-rw-r--r--   1 goossens et         104071 Jan 29 19:35 babelfrerr.ps

---------------------- FILE babelfrer.fstat -----------------------------
cmti10/parenright/t/i/S/m/o/d/parenleft/f/s/u/comma/P/l/a/n/two/c/p/e/r/acute/
cmff10/D/F/A/R/T/
cmr10/parenright/bracketright/g/t/i/v/D/Q/colon/x/F/S/m/one/fi/o/b/three/d/q/
      grave/parenleft/L/f/s/A/u/comma/C/P/nine/period/E/l/T/a/n/two/I/c/p/
      quoteright/four/bracketleft/X/e/r/acute/
cmbx10/g/t/x/S/m/one/U/fi/b/d/u/P/E/l/a/n/two/p/quoteright/e/r/
cmr12/t/i/v/x/m/cedilla/one/d/five/f/s/u/nine/j/E/l/a/n/two/c/p/quoteright/e/r/
cmtt10/g/t/i/braceleft/braceright/m/b/o/three/d/q/backslash/f/s/h/u/comma/w/
      period/l/equal/a/n/c/p/e/r/
cmmi10/period/
cmr7/o/A/e/
cmbx12/g/t/i/x/S/m/one/U/b/fi/d/grave/L/f/s/u/P/E/R/l/T/a/n/two/I/c/p/quoteright/e/r/acute/
```

C Example of using `fload` with the DC fonts

```
[1] dvips -hdraft.ps babelfre8 -o
This is dvipsk 5.58e Copyright 1986, 1994 Radical Eye Software
' TeX output 1995.01.29:1941' -> babelfre8.ps
<draft.ps><tex.pro><special.pro>. [1<colorcir.eps><tac2dim.eps>]
[2] dvips -hdraft.ps -PDC babelfre8 -o babelfre8dc.ps
This is dvipsk 5.58e Copyright 1986, 1994 Radical Eye Software
' TeX output 1995.01.29:1941' -> babelfre8dc.ps
<draft.ps><tex.pro><dcr12.pfb><dcbx12.pfb><dcbx10.pfb><dcr10.pfb><cmmi10.pfb>
```

```
<dctt10.pfb><dcr7.pfb><cmr10.pfb><cmr7.pfb><dcti10.pfb><texps.pro>
<special.pro>. [1<colorcir.eps><tac2dim.eps>]
[3] dvips -hdraft.ps -PDCr babelfre8 -o babelfre8r.ps
This is dvipsk 5.58e Copyright 1986, 1994 Radical Eye Software
' TeX output 1995.01.29:1941' -> babelfre8r.ps
<draft.ps><tex.pro><texps.pro><special.pro>. [1<colorcir.eps><tac2dim.eps>]
[4] ls -l babelfre8*.ps
-rw-r--r--   1 goossens et          60161 Jan 29 19:41 babelfre8.ps
-rw-r--r--   1 goossens et         720063 Jan 29 19:41 babelfre8dc.ps
-rw-r--r--   1 goossens et          28151 Jan 29 19:41 babelfre8r.ps
[5] fload babelfre8r.ps > babelfre8dcr.ps
Process PS file babelfre8r.ps.
Write font using statistic to babelfre8r.fstat
Process font 'dcr10' from file '/usr/local/lib/texmf/fonts/./dc/type1/dcr10.pfb'
Process font 'dcbx12' from file '/usr/local/lib/texmf/fonts/./dc/type1/dcbx12.pfb'
Process font 'dcti10' from file '/usr/local/lib/texmf/fonts/./dc/type1/dcti10.pfb'
Process font 'cmff10' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmff10.pfb'
Process font 'cmr10' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmr10.pfb'
Process font 'dctt10' from file '/usr/local/lib/texmf/fonts/./dc/type1/dctt10.pfb'
Process font 'dcbx10' from file '/usr/local/lib/texmf/fonts/./dc/type1/dcbx10.pfb'
Process font 'dcr12' from file '/usr/local/lib/texmf/fonts/./dc/type1/dcr12.pfb'
Process font 'dcr7' from file '/usr/local/lib/texmf/fonts/./dc/type1/dcr7.pfb'
Process font 'cmmi10' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmmi10.pfb'
Process font 'cmr7' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmr7.pfb'
Total of 12 files are processed.
[6] ls -l babelfre8dcr.ps
-rw-r--r--   1 goossens et         114515 Jan 29 19:42 babelfre8dcr.ps

---------------------- FILE babelfre8r.fstat -----------------------------
dcr10/Eacute/agrave/eacute/parenright/bracketright/g/t/i/v/D/Q/colon/x/F/S/m/
      one/fi/o/b/d/q/parenleft/L/f/s/A/u/comma/C/P/nine/period/E/l/T/a/n/two/
      I/c/p/quoteright/bracketleft/X/e/r/
dcbx12/egrave/eacute/g/t/i/x/S/m/one/U/b/fi/d/L/f/s/u/P/E/R/l/T/a/n/two/I/c/p/quoteright/e/r/
dcti10/eacute/parenright/t/i/S/m/o/d/parenleft/f/s/u/comma/P/l/a/n/two/c/p/e/r/
cmff10/D/F/A/R/T/
cmr10/one/three/two/four/
dctt10/g/t/i/braceleft/braceright/m/b/o/three/d/q/backslash/f/s/h/u/comma/w/
       period/l/equal/a/n/c/p/e/r/
dcbx10/g/t/x/S/m/one/U/fi/b/d/u/P/E/l/a/n/two/p/quoteright/e/r/
dcr12/ccedilla/t/i/v/x/m/one/d/five/f/s/u/nine/j/E/l/a/n/two/c/p/quoteright/e/r/
dcr7/A/
cmmi10/period/
cmr7/o/e/
```

D Example of using `fload` with a lot of mathematics

```
[21] fload mathexar.ps > mathexadcr.ps
Process PS file mathexar.ps.
Write font using statistic to mathexar.fstat
Process font 'cmmi7' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmmi7.pfb'
Process font 'dcr10' from file '/usr/local/lib/texmf/fonts/./dc/type1/dcr10.pfb'
Process font 'dcbx12' from file '/usr/local/lib/texmf/fonts/./dc/type1/dcbx12.pfb'
Process font 'cmr10' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmr10.pfb'
Process font 'cmsy7' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmsy7.pfb'
Process font 'dctt10' from file '/usr/local/lib/texmf/fonts/./dc/type1/dctt10.pfb'
Process font 'cmex10' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmex10.pfb'
Process font 'cmsy10' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmsy10.pfb'
Process font 'cmmi10' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmmi10.pfb'
Process font 'cmr7' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmr7.pfb'
```

```
Process font 'cmmi5' from file '/usr/local/lib/texmf/fonts/./cm/type1/cmmi5.pfb'
Total of 12 files are processed.
[3] ls -l mathexa*.ps
-rw-r--r--   1 goossens et         640749 Jan 29 19:12 mathexa.ps
-rw-r--r--   1 goossens et          87363 Jan 29 20:07 mathexadcr.ps
-rw-r--r--   1 goossens et           8748 Jan 29 19:13 mathexar.ps

---------------------- FILE matheaxr.fstat ----------------------------
cmmi7/gamma/k/A/l/n/
dcr10/udieresis/adieresis/germandbls/parenright/Z/g/t/B/i/v/hyphen/D/k/x/F/m/z/one/
      U/b/o/three/d/parenleft/f/s/A/N/h/u/comma/w/period/l/T/a/n/two/I/c/four/K/e/r/
dcbx12/udieresis/M/g/t/B/i/hyphen/k/F/S/m/one/o/three/d/f/s/A/h/u/l/a/n/two/c/p/e/r/
cmr10/parenright/six/plus/one/parenleft/equal/two/four/
cmsy7/minus/element/
dctt10/parenright/bracketright/g/t/i/m/b/o/d/q/parenleft/backslash/f/s/h/u/w/l/zero/
      a/n/c/p/bracketleft/e/r/
cmex10/parenrightbig/parenleftbigg/integraltext/summationdisplay/parenleftbig/
      integraldisplay/parenrightbigg/
cmsy10/minus/periodcentered/
cmmi10/k/x/z/b/d/f/comma/w/l/y/a/I/
cmr7/Gamma/one/three/two/four/
cmmi5/C/
```

E Using `fload` on the present article

```
[1] dvips -D1270 fload -ofload1270pk.ps
[2] dvips -D1270 -PCM fload -ofload1270cm.ps
[3] dvips -D1270 -PCMr fload -ofload1270cmr.ps
[4] fload fload1270cmr.ps fload1270cmp.ps
[5] ls fload1270*.ps
-rw-r--r--   1 goossens et         878948 Feb 12 16:26 fload1270cm.ps
-rw-r--r--   1 goossens et         501927 Feb 12 16:26 fload1270cmp.ps
-rw-r--r--   1 goossens et         262556 Feb 12 16:26 fload1270cmr.ps
-rw-r--r--   1 goossens et         838147 Feb 12 16:25 fload1270pk.ps
```

F Using `fload` with some $\mathcal{AMS}$ documents

```
[1] dvips -PCM -PAM amsldoc -o
This is dvipsk 5.58e Copyright 1986, 1994 Radical Eye Software
' TeX output 1995.02.16:1008' -> amsldoc.ps
<tex.pro><cmsy10.pfb><cmr17.pfb><cmr12.pfb><cmr10.pfb><cmr8.pfb><cmbx12.pfb>
<cmbx10.pfb><cmbsy10.pfb><cmbx7.pfb><cmr7.pfb><cmmi10.pfb><cmtt10.pfb>
<cmsy8.pfb><cmr6.pfb><cmbx8.pfb><cmti10.pfb><cmex10.pfb><cmtt8.pfb>
<cmtt12.pfb><cmmi7.pfb><cmsy7.pfb><cmr5.pfb><cmmi5.pfb><cmti8.pfb><cmsy5.pfb>
<cmmib10.pfb><cmbsy7.pfb><cmti7.pfb><cmmi8.pfb><texps.pro>. [1] [2] [3] [4]
[5] [6] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15]
[16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30]
[31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42] [43]
[2] ls -l amsldoc*.ps
-rw-r--r--   1 goossens et         488335 Feb 16 10:12 amsldocr.ps
-rw-r--r--   1 goossens et        1639085 Feb 16 10:13 amsldoc.ps
[3] ls -l testmath*.ps
-rw-r--r--   1 goossens et        1578903 Feb 16 10:29 testmath.ps
-rw-r--r--   1 goossens et        2032195 Feb 16 10:32 testmath1270pk.ps
-rw-r--r--   1 goossens et         181349 Feb 16 10:26 testmathnone.ps
-rw-r--r--   1 goossens et         480525 Feb 16 10:27 testmathr.ps
```

Exemple d'un article en français
29 janvier 1995

Table des matières

1 Une figure EPS 1
2 Exemple d'un tableau 1

Liste des figures

1 Deux images EPS 1

Liste des tableaux

1 Quelques commandes de l'option french de babel 1

1 Une figure EPS

Cette section montre comment inclure une figure PostScript[1] dans un document LaTeX. La figure 1 est insérée dans le texte à l'aide de la commande `\epsfig{file=colorcir.eps,width=3cm}`.

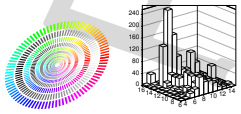


Figure 1: Deux images EPS

2 Exemple d'un tableau

Le tableau 1 à la page 1 montre l'utilisation de l'environnement `table`.

```
\primo 1°   \secundo 2°   \tertio 3°   \quatro 4°   2\ieme 2°
```

Tableau 1: Quelques commandes de l'option french de babel

Références

[1] Adobe Inc. *PostScript, manuel de référence (2e édition)* InterÉditions (France), 1992

Index

figure, 1
PostScript, 1
section, 1
index, 1
références, 1
tableau, 1

1

Figure 1: Example of page in French

In the figure above the word "DRAFT" was produced by using the following `draft.ps` header file with `dvips`:

```
% Prints "DRAFT" diagonally across the page
/bop-hook{gsave
    200 30 translate 65 rotate
    /cmff10 findfont 250 scalefont
    setfont 0 0 moveto
    0.85 setgray (DRAFT) show
  grestore} userdict begin def end
```

Note the use of the font `cmff10`. The presence of the five characters (corresponding to the word "DRAFT") of this font is picked up correctly by the font analysis step, as seen in Appendix B in the listing of the contents of the file `babelfref.stat`:

```
cmff10/D/F/A/R/T/
```

1 Mehrfachintegrale

`\iiint` und `\iiiint` erzeugen drei und vier Integralzeichen mit gleichmäßigem Zwischenraum, und zwar sowohl im normalen Text, wie z.B. $\iiint_A f(x,y,z)\,dx\,dy\,dz$ und $\iiiint_A f(w,x,y,z)\,dw\,dx\,dy\,dz$, als auch in abgesetzten Formeln.

$$\iiint_A f(x,y,z)\,dx\,dy\,dz \tag{1}$$

$$\iiiint_A f(w,x,y,z)\,dw\,dx\,dy\,dz \tag{2}$$

2 Binomische Ausdrücke

Für binomische Ausdrücke, wie etwa $\binom{n}{k}$ gibt es die Befehle `\binom`, `\dbinom` und `\tbinom`. `\binom` ist eine Kurzform für `\fracwithdelims()[0pt]`.

$$\begin{aligned}\sum_{\gamma\in\Gamma_C} I_\gamma &= 2^k - \binom{k}{1}2^{k-1} + \binom{k}{2}2^{k-2} \\ &\quad + \cdots + (-1)^l\binom{k}{l}2^{k-l} + \cdots + (-1)^k \\ &= (2-1)^k = 1\end{aligned} \tag{3}$$

3 Split-Formeln

Die `split`-Umgebung erzeugt keine Numerierung, da sie nur innerhalb anderer abgesetzter Formelumgebungen verwendet werden kann, wie etwa `equation`, `align` oder `gather`. Die Formelnummer wird dann von der äußeren Umgebung erzeugt.

$$\begin{aligned}(a+b)^4 &= (a+b)^2(a+b)^2 \\ &= (a^2+2ab+b^2)(a^2+2ab+b^2) \\ &= a^4+4a^3b+6a^2b^2+4ab^3+b^4\end{aligned} \tag{4}$$

Figure 2: `dvips` output (DC and CM math fonts)

Note added in Proof

Sergey Lesenko has implemented partial embedding of Type 1 font sources into PostScript files within the framework of Tom Rokicki's `dvips` program. This work is presented in an article in the proceedings of the 1995 TUG Conference in St. Petersburg (*TUGboat* 16(3), *to be published*). Lysenko is working closely with Tom Rokicki to integrate his code into the next major release of `dvips`, so that it is anticipated that for TeX users, partial font embedding will be a trivial task in the near future.

◇ Basil K. Malyshev
IHEP, Protvino, Russia
Email: `goossens@cern.ch`

◇ Michel Goossens
CERN, Geneva, Switzerland
Email: `malyshev@desert.ihep.su`

Tight setting with TEX

Alan Jeffrey

1 Introduction

This note describes some experiments with setting text matter in TEX using Adobe Times, which is a very tightly spaced text font. Acceptable results are possible with TEX, but some tweaking is required.

2 Setting text

Here is some text set in Computer Modern:

> On November 14, 1885, Senator & Mrs. Leland Stanford called together at their San Francisco mansion the 24 prominent men who had been chosen as the first trustees of The Leland Stanford Junior University. They handed to the board the Founding Grant of the University, which they had executed three days before. This document—with various amendments, legislative acts, and court decrees—remains as the University's charter. In bold, sweeping language it stipulates that the objectives of the University are "to qualify students for personal success and direct usefulness in life; and to promote the publick welfare by exercising an influence in behalf of humanity and civilization, teaching the blessings of liberty regulated by law, and inculcating love and reverence for the great principles of government as derived from the inalienable rights of man to life, liberty, and the pursuit of happiness."

And here is the same text set in Adobe Times:

> On November 14, 1885, Senator & Mrs. Leland Stanford called together at their San Francisco mansion the 24 prominent men who had been chosen as the first trustees of The Leland Stanford Junior University. They handed to the board the Founding Grant of the University, which they had executed three days before. This document—with various amendments, legislative acts, and court decrees—remains as the University's charter. In bold, sweeping language it stipulates that the objectives of the University are "to qualify students for personal success and direct usefulness in life; and to promote the publick welfare by exercising an influence in behalf of humanity and civilization, teaching the blessings of liberty regulated by law, and inculcating love and reverence for the great principles of government as derived from the inalienable rights of man to life, liberty, and the pursuit of happiness."

The first thing I can see about these two texts is how much darker Adobe Times is—partially this is because Computer Modern is a very light, brilliant face, but partially it's because Adobe's Times is a very dark cut, of somewhat dubious character.[1]

The next point of note is that the Times setting is two lines shorter. This economy of space is one of the main reasons for publishers selecting Times as a book font!

Looking at the Computer Modern setting, it is very variably set: the difference between the setting of tight and loose lines is very high, and is much less so with the Times setting, which is generally much tighter. In fact, we can make the setting of Times tighter still, by appropriate settings of the TEX paragraph parameters:

> On November 14, 1885, Senator & Mrs. Leland Stanford called together at their San Francisco mansion the 24 prominent men who had been chosen as the first trustees of The Leland Stanford Junior University. They handed to the board the Founding Grant of the University, which they had executed three days before. This document—with various amendments, legislative acts, and court decrees—remains as the University's charter. In bold, sweeping language it stipulates that the objectives of the University are "to qualify students for personal success and direct usefulness in life; and to promote the publick welfare by exercising an influence in behalf of humanity and civilization, teaching the blessings of liberty regulated by law, and inculcating love and reverence for the great principles of government as derived from the inalienable rights of man to life, liberty, and the pursuit of happiness."

The settings which achieved this were:

```
\frenchspacing
\leftskip=0pt minus 1pt
\rightskip=0pt minus 1pt
\hfuzz=0pt
\tolerance=800
\emergencystretch=0pt
\doublehyphendemerits=2500
```

Or, in English:

- No extra space after punctuation, for example "by law, and".
- Up to 2pt shrink in the line width, which allows tight lines to have up to 1pt of "wadding" added at the left and right.
- No overfull `\hbox`es allowed, since the wadding compensates.

[1] The settings of Times described are with the older metrics (pre mid-1995).

- Less tolerance of underfull boxes.
- No extra space for underfull boxes.
- Two hyphenated lines in a row aren't too bad.

In producing these settings, I realized that Knuth allowed TEX a lot of flexibility about producing underfull boxes, but very little about producing overfull boxes, without jutting out into the margin. An `\emergencyshrink` would be very useful! In addition, some typesetters would allow double hyphenated lines but not triple hyphenated lines, but TEX has no way of specifying that.

One unfortunate result of these settings is the three stacked occurrences of 'the' at the end of the paragrapgh, especially since the last one has noticably less wadding than the first and second. A bit of hand-correction produces what I think is probably the best achievable setting with standard Adobe Times:

> On November 14, 1885, Senator & Mrs. Leland Stanford called together at their San Francisco mansion the 24 prominent men who had been chosen as the first trustees of The Leland Stanford Junior University. They handed to the board the Founding Grant of the University, which they had executed three days before. This document—with various amendments, legislative acts, and court decrees—remains as the University's charter. In bold, sweeping language it stipulates that the objectives of the University are "to qualify students for personal success and direct usefulness in life; and to promote the publick welfare by exercising an influence in behalf of humanity and civilization, teaching the blessings of liberty regulated by law, and inculcating love and reverence for the great principles of government as derived from the inalienable rights of man to life, liberty, and the pursuit of happiness."

However, for those willing to brave virtual fonts, one last improvement can be achieved. The hyphen character in Adobe Times has large sidebearings, which can be reduced by an appropriate virtual font. This produces:

> On November 14, 1885, Senator & Mrs. Leland Stanford called together at their San Francisco mansion the 24 prominent men who had been chosen as the first trustees of The Leland Stanford Junior University. They handed to the board the Founding Grant of the University, which they had executed three days before. This document—with various amendments, legislative acts, and court decrees—remains as the University's charter. In bold, sweeping language it stipulates that the objectives of the University are "to qualify students for personal success and direct usefulness in life; and to promote the publick welfare by exercising an influence in behalf of humanity and civilization, teaching the blessings of liberty regulated by law, and inculcating love and reverence for the great principles of government as derived from the inalienable rights of man to life, liberty, and the pursuit of happiness."

The Cork fonts include separate hyphen characters for in-line hyphenation (for example in the word 'in-line') and for line-breaking hyphenation. It may be that this can be exploited to produce more beautiful pages with PostScript fonts.

3 Setting math

Now we can play the same game with some mathematics. Here's some Computer Modern mathematics (shown ragged right, to make it easier to see what's going on):

> Suppose $f \in \mathcal{S}_n$ and $g(x) = (-1)^{|\alpha|}x^\alpha f(x)$. Then $g \in \mathcal{S}_n$; now (c) implies that $\hat{g} = D_\alpha \hat{f}$ and $P \cdot D_\alpha \hat{f} = P \cdot \hat{g} = (P(D)g)\hat{}$, which is a bounded function, since $P(D)g \in L^1(R^n)$. This proves that $\hat{f} \in \mathcal{S}_n$. If $f_i \to f$ in $\mathcal{S}_n$, then $f_i \to f$ in $L^1(R^n)$. Therefore $\hat{f}_i(t) \to \hat{f}(t)$ for all $t \in R^n$. That $f \to \hat{f}$ is a *continuous* mapping of $\mathcal{S}_n$ into $\mathcal{S}_n$ follows now from the closed graph theorem. *Functional Analysis*, W. Rudin, McGraw–Hill, 1973.

And again in Times, using the mathptm math fonts:

> Suppose $f \in \mathcal{S}_n$ and $g(x) = (-1)^{|\alpha|}x^\alpha f(x)$. Then $g \in \mathcal{S}_n$; now (c) implies that $\hat{g} = D_\alpha \hat{f}$ and $P \cdot D_\alpha \hat{f} = P \cdot \hat{g} = (P(D)g)\hat{}$, which is a bounded function, since $P(D)g \in L^1(R^n)$. This proves that $\hat{f} \in \mathcal{S}_n$. If $f_i \to f$ in $\mathcal{S}_n$, then $f_i \to f$ in $L^1(R^n)$. Therefore $\hat{f}_i(t) \to \hat{f}(t)$ for all $t \in R^n$. That $f \to \hat{f}$ is a *continuous* mapping of $\mathcal{S}_n$ into $\mathcal{S}_n$ follows now from the closed graph theorem. *Functional Analysis*, W. Rudin, McGraw–Hill, 1973.

And again with tighter setting:

> Suppose $f \in \mathcal{S}_n$ and $g(x) = (-1)^{|\alpha|}x^\alpha f(x)$. Then $g \in \mathcal{S}_n$; now (c) implies that $\hat{g} = D_\alpha \hat{f}$ and $P \cdot D_\alpha \hat{f} = P \cdot \hat{g} = (P(D)g)\hat{}$, which is a bounded function, since $P(D)g \in L^1(R^n)$. This proves that $\hat{f} \in \mathcal{S}_n$. If $f_i \to f$ in $\mathcal{S}_n$, then $f_i \to f$ in $L^1(R^n)$. Therefore $\hat{f}_i(t) \to \hat{f}(t)$ for all $t \in R^n$. That $f \to \hat{f}$ is a *continuous* mapping of $\mathcal{S}_n$ into $\mathcal{S}_n$ follows now from the closed graph theorem. *Functional Analysis*, W. Rudin, McGraw–Hill, 1973.

Suppose $f \in \mathcal{S}_n$ and $g(x) = (-1)^{|\alpha|} x^\alpha f(x)$. Then $g \in \mathcal{S}_n$; now (c) implies that $\hat{g} = D_\alpha \hat{f}$ and $P \cdot D_\alpha \hat{f} = P \cdot \hat{g} = (P(D)g)\hat{}$, which is a bounded function, since $P(D)g \in L^1(R^n)$. This proves that $\hat{f} \in \mathcal{S}_n$. If $f_i \to f$ in $\mathcal{S}_n$, then $f_i \to f$ in $L^1(R^n)$. Therefore $\hat{f}_i(t) \to \hat{f}(t)$ for all $t \in R^n$. That $f \to \hat{f}$ is a *continuous* mapping of $\mathcal{S}_n$ into $\mathcal{S}_n$ follows now from the closed graph theorem. *Functional Analysis*, W. Rudin, McGraw–Hill, 1973.

Suppose $f \in \mathcal{S}_n$ and $g(x) = (-1)^{|\alpha|} x^\alpha f(x)$. Then $g \in \mathcal{S}_n$; now (c) implies that $\hat{g} = D_\alpha \hat{f}$ and $P \cdot D_\alpha \hat{f} = P \cdot \hat{g} = (P(D)g)\hat{}$, which is a bounded function, since $P(D)g \in L^1(R^n)$. This proves that $\hat{f} \in \mathcal{S}_n$. If $f_i \to f$ in $\mathcal{S}_n$, then $f_i \to f$ in $L^1(R^n)$. Therefore $\hat{f}_i(t) \to \hat{f}(t)$ for all $t \in R^n$. That $f \to \hat{f}$ is a *continuous* mapping of $\mathcal{S}_n$ into $\mathcal{S}_n$ follows now from the closed graph theorem. *Functional Analysis*, W. Rudin, McGraw–Hill, 1973.

Suppose $f \in \mathcal{S}_n$ and $g(x) = (-1)^{|\alpha|} x^\alpha f(x)$. Then $g \in \mathcal{S}_n$; now (c) implies that $\hat{g} = D_\alpha \hat{f}$ and $P \cdot D_\alpha \hat{f} = P \cdot \hat{g} = (P(D)g)\hat{}$, which is a bounded function, since $P(D)g \in L^1(R^n)$. This proves that $\hat{f} \in \mathcal{S}_n$. If $f_i \to f$ in $\mathcal{S}_n$, then $f_i \to f$ in $L^1(R^n)$. Therefore $\hat{f}_i(t) \to \hat{f}(t)$ for all $t \in R^n$. That $f \to \hat{f}$ is a *continuous* mapping of $\mathcal{S}_n$ into $\mathcal{S}_n$ follows now from the closed graph theorem. *Functional Analysis*, W. Rudin, McGraw–Hill, 1973.

Figure 1: Math in Computer Modern, Adobe Times, and tight Adobe Times

The additional math parameters are taken from the mathptm LaTeX 2_ε package:

```
\thinmuskip=2mu
\medmuskip=2.5mu plus 1mu minus 1mu
\thickmuskip=4mu plus 1.5mu minus 1mu
```

It's impossible to show the effects of tight text and math setting in the narrow measure of a *TUGboat* quotation, so in Figure 1 you can see the effect of Computer Modern, Adobe Times, and tight Adobe Times flush right.

It's worth noting that there's very little difference between the loose and tight settings of Adobe Times, becuase there are very few good linebreaks for mathematically heavy material, so the only difference is that the text stretches more and the math stretches a bit less.

Acknowledgements

This note was inspired by a talk by Richard Southall at TUG 93, in which he described the problems with setting tight text with TeX and Computer Modern. I hope this note shows that acceptable results are possible with TeX, as long as care is taken with fonts and setting parameters. However, as Richard Southall pointed out, TeX is much more prone to loose setting than to tight.

◇ Alan Jeffrey
School of Cognitive and Computing Sciences
University of Sussex
Falmer
Brighton
BN1 9QH
UK
Email: `alanje@cogs.susx.ac.uk`

XΥMTEX for Drawing Chemical Structural Formulas

Shinsaku Fujita

Abstract

XΥMTEX,[1] a macro package of combined LATEX style files, has been developed for drawing a wide variety of chemical structural formulas. The commands of XΥMTEX have a set of systematic arguments for specifying substituents and their positions, endocyclic double bonds, and bond patterns. In some cases, they have an additional argument for specifying hetero-atoms on the vertices of heterocycles. As a result of this systematic feature, XΥMTEX works effectively as a practical tool within the "device-independent" concept of TEX.

1 Introduction

A few years ago, in order to expand the use of TEX into various fields of chemistry, I decided to write a book [1] that would use LATEX style files and BIBTEX `bst` files suitable for such scientific journals as *Journal of the American Chemical Society*, *Science*, and *Nature* (potentially there were some thirty journals in the field which might benefit). However, while preparing the book I encountered difficulties in introducing methods for drawing formulas of chemical structures. Although I had dealt with ChemTEX [2], epic [3] and PICTEX [4] as device-independent methods, as well as with PostScript [5] and tpic [6] as

[1] The Xs of XΥMTEX should be pronounced as a Greek chi or simply as 'k' in 'kyumtek'.

device-dependent methods, it was difficult to recommend any one of them as a standard method.

ChemTEX typesets structural formulas of high quality in a device-independent manner. Its commands, however, should be replaced by more systematic ones in order to cover structures with a wide range of substitution. As for epic and PICTEX, in themselves they have no facilities for drawing chemical structures. Moreover, they produce output of lesser quality than ChemTEX, especially in printing chemical bonds. Among device-dependent methods, encapsulated PostScript is now recognized as the predominant method since chemical structural formulas are usually drawn with tools whose storing and printing processes are based on PostScript. However, it is still desirable to develop a convenient method according to the device-independent concept, since this is a fundamental philosophy in TEX typesetting, and encourages electronic submission and exchange of information.

2 Features of XymTEX

Therefore, it seemed necessary to take all of these issues into consideration and devise something new. XymTEX was developed as a device-independent method with systematic commands (control sequences) for drawing structural formulas [7]. The features and advantages of XymTEX are summarized below:

1. The name XymTEX is the uppercase form of $\chi\upsilon\mu\tau\epsilon\chi$, in which $\chi\upsilon\mu$ is the Greek counterpart of the stem 'chem' of 'chemistry'. When the logo XymTEX is unavailable, you should type `XyMTeX`.
2. XymTEX requires the LATEX `picture` environment only, ensuring portability (since LATEX is part of most TEX distributions). Thus, wide adaptations for personal computers are available and a variety of printers can be used as output devices.
3. XymTEX should be used within a large version of LATEX.
4. Structural formulas written with XymTEX produce high-quality output, since they use LATEX fonts.
5. Each command name corresponds to a master template to be drawn. It can be easily remembered, since it stems from the familiar nomenclature of organic compounds.
6. The invariant part of a structure (the master template containing fixed bonds and atoms) is automatically printed with no designation.
7. The variant parts of a structure (substituents, additional bonds and atoms) are designated by up to four arguments: SUBSLIST, OPT, BONDLIST, and ATOMLIST.
8. Substituents and their positions are given by a single argument (SUBSLIST) in which they are listed consecutively, with semicolons as delimiters. It follows that an arbitrary number of substituents can be written in the SUBSLIST.
9. A command of frequent occurrence has an optional argument (OPT) of one or two characters for showing a pattern of bonds or aromatization.
10. Additional endocyclic bonds are designated by an optional argument (BONDLIST) in which one character corresponds to each of the bonds.
11. A more general command for drawing heterocycles takes an additional argument (ATOMLIST), so that a set of hetero-atoms are typeset on the vertices of the master template after truncation of edges.
12. Commands with a common stem but different suffixes ('v', 'vi', 'h', 'hi') are provided for drawing the same structure in different ways.
13. Each structure created by a XymTEX command is regarded as a letter, or more exactly, as a TEX box. Thus, it is controlled by the inherent mechanism of TEX in breaking paragraphs (containing such structures) into lines as well as in making the lines into pages.
14. The recognition of a XymTEX structure as a TEX box permits us to use XymTEX commands in various LATEX environments such as `center`, `equation` and `tabular`.
15. In extreme cases, a XymTEX command can be used in the argument of another command. For example, it may be utilized in the argument of `\section` in conjunction with the `\protect` command. However, such modes of usage should not be recommended to regular users, since they may cause unexpected errors.

3 Drawing Benzene and Naphthalene Derivatives

Let us first draw benzene and naphthalene derivatives as examples. XymTEX contains nine style files for drawing various categories of chemical structural formulas (Table 1). Since the macros we will require are stored in `carom.sty`, the document file begins as follows:

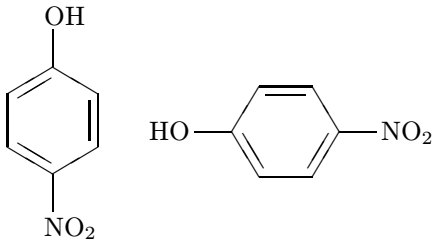
```
\documentstyle[epic,carom]{article}
\begin{document}
(body)
\end{document}
```

where `epic.sty` is also included in any order as an option for drawing dotted lines. This is the usual form of LaTeX documents.

The constitution of a command is quite simple. To draw the structure of 4-nitrophenol, you write the following simple statements in the body of your document:

```
\bzdrv{1==OH;4==NO$_2$}
\bzdrh{1==HO;4==NO$_2$}
```

where each argument is a SUBSLIST, listing substituents with their bonds. This generates the following:



A semicolon separates each mode of substitution, where a double equality symbol (==) is used as a delimiter between a substitution position and a substituent. Thus, the two arguments state that position 1 takes a hydroxyl group through a single bond ('1==OH') and position 4 takes a nitro group through a single bond ('4==NO_2'), where each single bond is automatically drawn without explicit declaration. Since statements in SUBSLIST arguments follow the nomenclature of organic compounds, as shown in these examples, most organic chemists and secretaries with appropriate training can write them down easily. The suffixes 'v' and 'h' generally indicate vertical and horizontal forms of printed formulas.

To draw three structures with different aromatic expressions for 1-bromo-4-chlorobenzene, you use `\bzdrv[OPT]{SUBSLIST}` as follows:

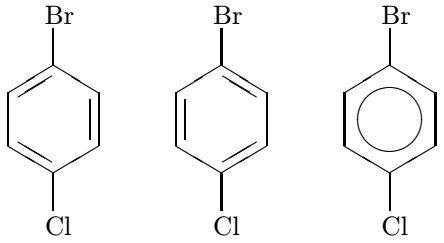
```
\bzdrv[r]{1==Br;4==Cl}
\bzdrv[l]{1==Br;4==Cl}
\bzdrv[c]{1==Br;4==Cl}
```

where the letters in brackets are optional arguments for representing patterns of double bonds. The standard mode of displaying alternant double bonds (to the right-hand side of the diagram) is the default (e.g., when no optional argument is used, as in the first example); specifying 'r' in square brackets will also yield the right-handed mode. The letter 'l' in brackets generates an alternative (left-handed) mode of alternant double bonds; 'c' in square brackets expresses an inner circle. As a result, you get

Table 1: Style Files in XyMTeX

file name	printed structures
`aliphat.sty`	aliphatic compounds
`carom.sty`	vertical and horizontal types of cyclic compounds
`ccycle.sty`	bicyclic compounds etc.
`chemstr.sty`	basic commands
`hcycle.sty`	pyranoses and furanoses
`hetarom.sty`	vertical types of heterocyclic compounds
`hetaromh.sty`	horizontal types of heterocyclic compounds
`locant.sty`	locant numbers
`lowcycle.sty`	five-or-less-membered carbocycles

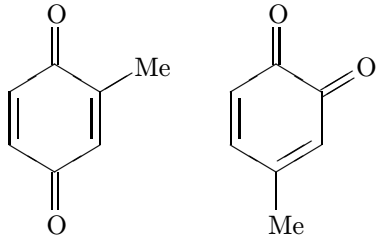
three structural formulas with different bond patterns for the same compound:



The `\bzdrv` command is also used to typeset *p*- and *o*-benzoquinone derivatives. Thus, the code:

```
\begin{center}
\bzdrv[p]{1D==O;4D==O;2==Me}
\bzdrv[o]{1D==O;2D==O;4==Me}
\end{center}
```

produces



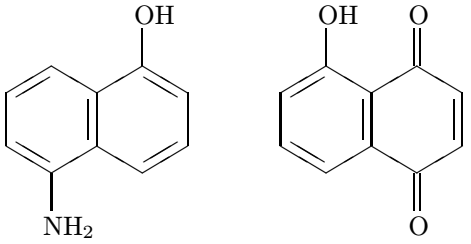
where the optional arguments 'p' and 'o' control patterns of endocyclic double bonds. The carbonyl double bonds are designated by means of a bond modifier 'D' coupled with a preceding locant number; thus, the string '1D==O' represents an oxygen atom through an exocyclic double bond.

Naphthalenes and naphthoquinones are typeset by using the `\naphdrv` command and so on. For example, the code:

```
\begin{center}
```

```
\naphdrv{1==OH;5==NH$_2$}
\naphdrv[p]{1D==O;4D==O;8==OH}
\end{center}
```

prints the following fused structures:



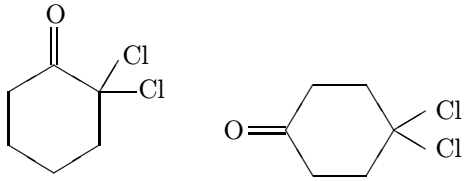
The `\naphdrv` command also takes optional arguments ('oa' to 'of') in order to print all possible structures of *o*-naphthoquinones. The command is also capable of drawing other naphthoquinones such as 2,6-naphthoquinones.

4 Drawing Cyclohexane Derivatives

The command `\cyclohexanev` and related ones are used to typeset cyclohexane derivatives. These commands are also contained in `carom.sty`. They are capable of drawing geminal substituents by using appropriate bond modifiers such as 'Sa' and 'Sb'. For example, the code:

```
\begin{center}
\cyclohexanev{1D==O;2Sa==Cl;2Sb==Cl}
\cyclohexaneh{1D==O;4Sa==Cl;4Sb==Cl}
\end{center}
```

produces the following structures:

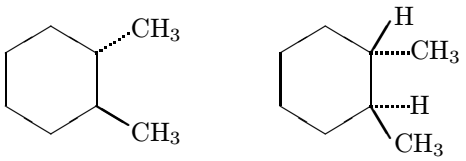


Note that the suffixes 'v' and 'h' are in accord with the general convention described above.

For specifying the stereochemistries of cyclohexanes more explicitly, we use bond modifiers of single type ('A' and 'B') as well as those of geminal type ('SA' and 'SB'). For example,

```
\begin{center}
\cyclohexanev{2A==CH$_3$;3B==CH$_3$}
\cyclohexanev{2SA==CH$_3$;2SB==H;%
 3SB==CH$_3$;3SA==H}
\end{center}
```

yields the following alternative expressions of *trans*-1,2-dimethylcyclohexane:

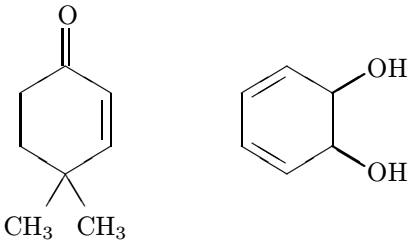


where the α-bonds ('A' and 'SA') are represented by dotted lines and the β-bonds ('B' and 'SB') are printed with boldfaced lines.

Commands such as `\cyclohexanev` take an optional argument BONDLIST that contains one or more letters ('a' to 'f') for designating endocyclic double bonds in a bond-by-bond fashion. Thus, we use `\cyclohexanev[BONDLIST]{SUBSLIST}`. For example,

```
\begin{center}
\cyclohexanev[b]{1D==O;%
 4Sa==CH$_3$;4Sb==CH$_3$}
\cyclohexanev[df]{2B==OH;3B==OH}
\end{center}
```

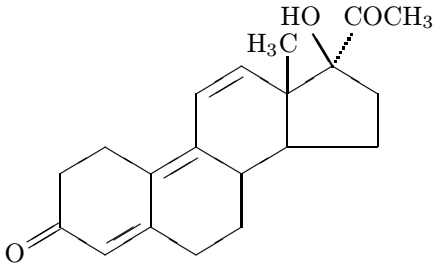
generates the following structural formulas with endocyclic double bonds:



XeTeX is capable of drawing more complicated structures such as steroids in a similar way. Let us write the code:

```
\begin{center}
\steroid[dim]{3D==O;%
 {{13}B}==\lmoiety{H$_3$C};{{17}SB}==HO;%
 {{17}SA}==COCH$_3$}
\end{center}
```

where the optional argument 'dim' gives three endocyclic double bonds, giving us a steroid derivative:



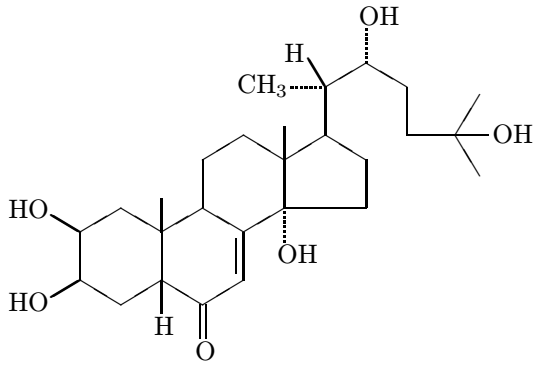
the methyl group of which attaches to the fused 13-position in such a manner that the right terminal

carbon of the methyl is linked to the corresponding bond by using the command `\lmoiety`.

The command `\steroidchain` is used to draw an insect hormone α-ecdysone just by designating a set of substituents in the SUBSLIST argument. Thus, we write the code:

```
\begin{center}
\unitlength.09pt
\steroidchain[g]{%
 2B==HO;3B==HO;{{10}B}==;5B==H;%
 6D==O;{{13}B}==;{{14}A}==OH;%
 {{20}SA}==CH$_3$;{{20}SB}==H;%
 {{22}A}==OH;{{25}}==H}
\end{center}
```

in which empty substituents are allowed to show implicit methyl groups at the 10- and 13-positions. This results in the following complex formula. In this example, I have slightly reduced the size of the structural formula by setting `\unitlength.09pt`— the default setting in XeTeX for unit length is 0.1pt.



It should be noted that further reduction of sizes is usually unsuccessful because the LaTeX `picture` environment is incapable of drawing lines of short lengths and of arbitrary slopes.

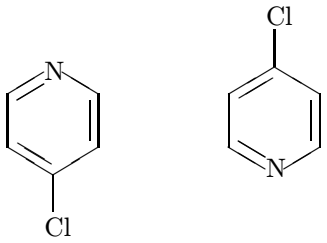
In the future, this restriction concerning line lengths and slopes should be overcome so that one could draw chemical structures more conveniently. XeTeX would then become an automatic tool linked with chemical drawing software. This linkage would mean that XeTeX codes could be created automatically in the future rather than manually as in the present situation. The future aim for XeTeX is therefore to implement a device-independent method whose codes would be created automatically with some kind of graphical user interface.

5 Drawing Heterocycles

For the purpose of drawing heterocyclic compounds, `\documentstyle` must contain `hetarom.sty` in the optional argument. Let us draw pyridine derivatives, using the following code:

```
\begin{center}
\pyridinev{4==Cl}
\pyridinevi{4==Cl}
\end{center}
```

This yields two pyridine structures of inverse types:

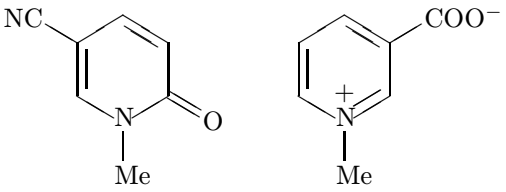


To show a ring nitrogen at the top of the pryidine derivative, the suffix 'v' is used; to mark the presence of a nitrogen atom at the bottom position of the ring, 'vi' is used. As a result, the corresponding positions in the alternative structures have common locant numbers, starting from the respective nitrogen atoms; thus, the 4-chloropyridine has been drawn in two ways without any changes to the SUBSLIST argument. This convention is also applied to commands for drawing other heterocycles.

The `\pyridinev` and related commands take an optional argument BONDLIST to designate endocyclic double bonds other than the default settings for alternant double bonds. Thus, we use `\pyridinev[BONDLIST]{SUBSLIST}`, in which one or more characters selected from 'a' to 'f' are involved in the BONDLIST. For example, the code:

```
\begin{center}
\pyridinevi[ce]{2D==O;1==Me;5==NC}
\pyridinevi[ace{1+}]{1==Me;3==COO$^{-}$}
\end{center}
```

produces the following structures:

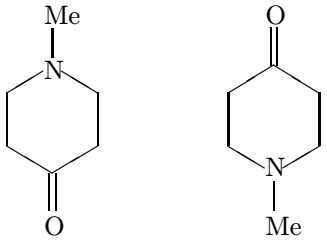


Note that the BONDLIST argument in the latter example contains a descriptor '1+' for denoting a plus charge on the ring nitrogen. Such a descriptor consists of a locant number and a character to be printed, both of which are bundled with braces according to the TeX grammar.

The default pattern of endocyclic double bonds for drawing six-membered heterocycles is an alternant pattern to complete their aromaticity. If you intend to typeset saturated heterocycles, you should give an empty optional argument:

```
\begin{center}
\pyridinev[]{4D==O;1==Me}
\pyridinevi[]{4D==O;1==Me}
\end{center}
```

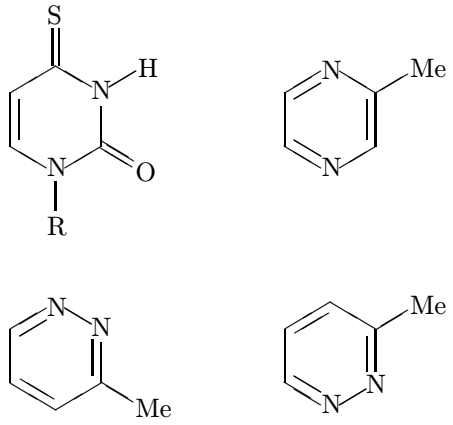
which yields the following structures:

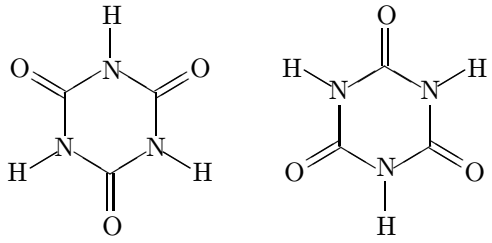


In order to show a wide variety of commands for drawing heterocyclic compounds, let us test the following statements:

```
\begin{center}
\pyrimidinevi[e]{%
 1==R;3==H;2D==O;4D==S} \qquad
\pyrazinev{2==Me} \\
\pyridazinev{3==Me} \qquad
\pyridazinevi{3==Me} \\
\triazinev[]{2D==O;4D==O;%
 6D==O;1==H;3==H;5==H} \qquad
\triazinevi[]{2D==O;4D==O;%
 6D==O;1==H;3==H;5==H}
\end{center}
```

Note that the pair of codes before each \\ constructs a text line during LaTeX processing, since XeTeX views each structure as a letter (or a TeX box). The following heterocycles result:

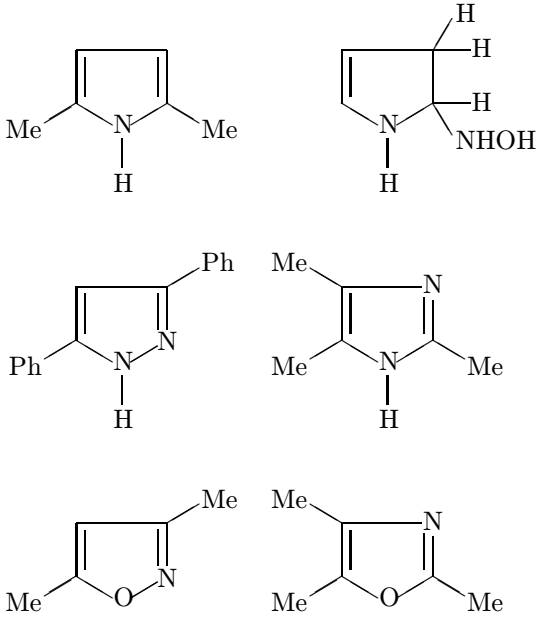




Since 5-membered heterocycles comprise a predominant family of organic compounds, XeTeX has versatile facilities for drawing them:

```
\begin{center}
\pyrrolev{1==H;2==Me;5==Me}
\pyrrolev[d]{1==H;2Sa==H;%
 2Sb==NHOH;3Sa==H;3Sb==H} \\
\pyrazolev{1==H;3==Ph;5==Ph}
\imidazolev{1==H;2==Me;4==Me;5==Me} \\
\isoxazolev{3==Me;5==Me}
\oxazolev{2==Me;4==Me;5==Me}
\end{center}
```

The command names come from those of master templates, i.e., pyrrole, pyrazole, imidazole, isoxazole, and oxazole. As before, the suffix conventions ('v', 'vi', 'h' and 'hi') are also effective in these commands. The above code yields a variety of 5-membered cyclic compounds:

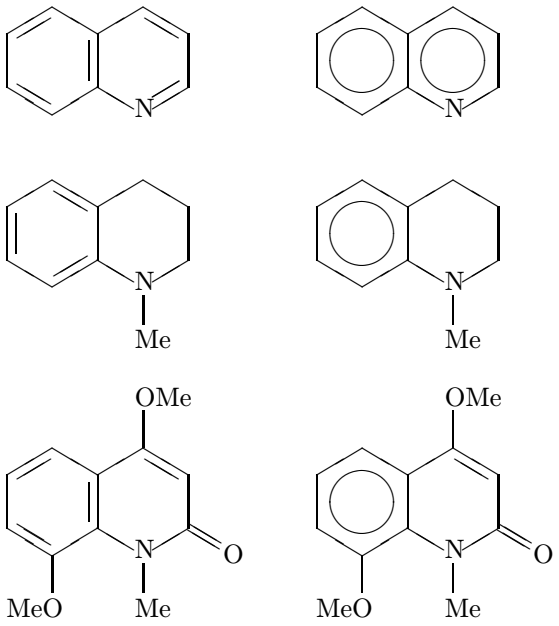


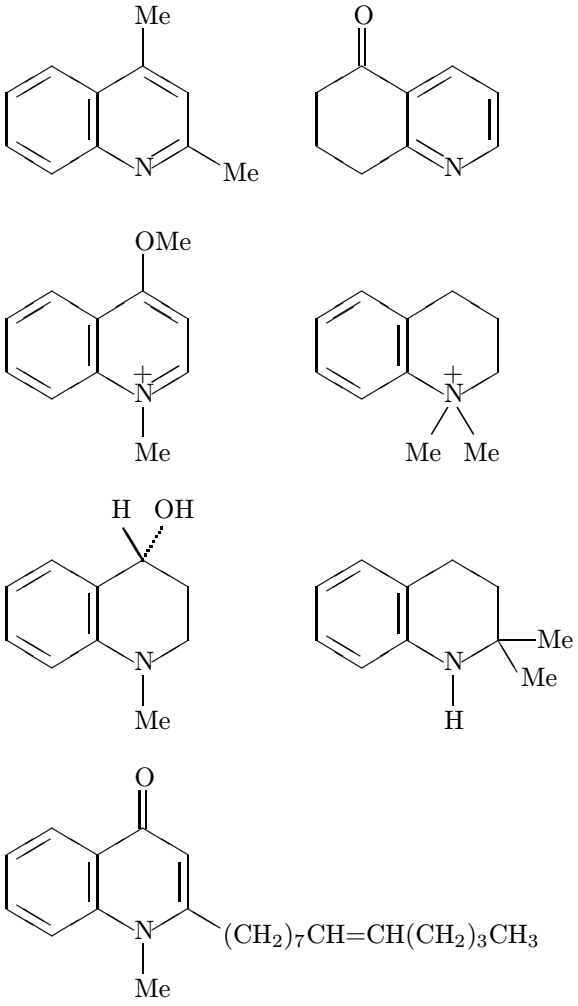
XeTeX is capable of also drawing fused heterocyclic compounds. These compounds can take a wide variety of substitutions and bond patterns.

Every command in XeLaTeX has therefore been designed to be able to deal with such diversity by using BONDLIST and SUBSLIST arguments. To illustrate the flexibility of XeLaTeX commands, let us test `\quinolinevi` in various situations:

```
\begin{center}
\quinolinevi{}
\quinolinevi[AB]{} \\[-16pt]
\quinolinevi[egi]{1==Me}
\quinolinevi[A]{1==Me} \\[16pt]
\quinolinevi[cfhk]{1==Me;2D==O;%
 4==OMe;8==\lmoiety{MeO}}
\quinolinevi[Ac]{1==Me;2D==O;%
 4==OMe;8==\lmoiety{MeO}} \\[16pt]
\quinolinevi{2==Me;4==Me}
\quinolinevi[bdj]{5D==O} \\
\quinolinevi[r{1+}]{1==Me;4==OMe}
\quinolinevi[fhk{1+}]{%
 1Sa==Me;1Sb==Me} \\[16pt]
\quinolinevi[fhk]{1==Me;4SA==OH;4SB==H}
\quinolinevi[fhk]{%
 1==H;2Sa==Me;2Sb==Me} \\[16pt]
\quinolinevi[bfhk]{1==Me;4D==O;%
 2==(CH$_2$)$_7$CH=CH(CH$_2$)$_3$CH$_3$}
\phantom{\quinolinevi{}}
\end{center}
```

This gives us the following test structures:





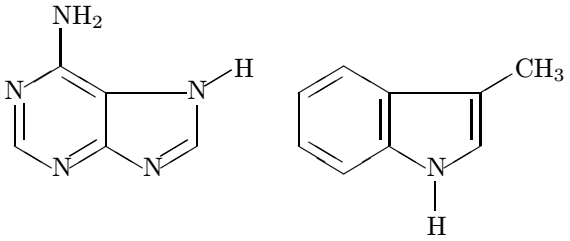
This last example contains a long-chain substituent with a double bond represented by a single equality symbol (=). For this reason, we use double equality (==) as a delimiter in the SUBSLIST argument.

XeLaTeX includes other commands for drawing fused heterocycles with two 6-membered rings: isoquinolines, quinoxalines, quinazolines, cinnolines, and pteridines. These commands can be used in the same way as described for the `\quinolinevi` command, where 'v', 'vi', 'h', and 'hi' are also effective.

XeLaTeX has various commands for drawing heterocycles with fused 5- and 6-membered rings: purines, indoles, indolizines, isoindoles, benzofuranes, isobenzofuranes, and benzoxazoles. For example, adenine and 3-methylindole are drawn by using the following code:

```
\begin{center}
\purinev[adfh]{3==H;4==NH$_2$}
\indolev{1==H;3==CH$_3$}
\end{center}
```

which gives the following results:

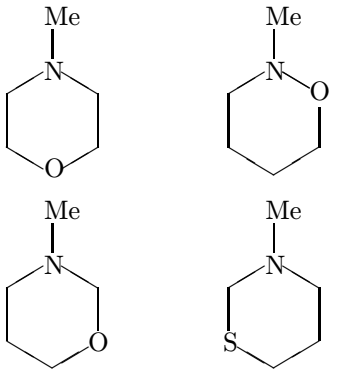


In the preceding paragraphs of this section, each mother skeleton is associated with a specific command. This approach is combinatorially explosive in nature since there are further categories of heterocyclic master templates. More general commands have therefore been designed to have the additional function of specifying inner atoms on rings, as in `\sixheterovi[BONDLIST]{ATOMLIST}{SUBSLIST}`.

Each of the following code examples contains two arguments in addition to an empty optional BONDLIST argument. Among them, the second argument is an ATOMLIST for specifying the positions and species of hetero-atoms.

```
\begin{center}
\sixheterovi[]{1==O;4==N}{4==Me}
\sixheterovi[]{3==O;4==N}{4==Me} \\
\sixheterovi[]{2==O;4==N}{4==Me}
\sixheterovi[]{6==S;4==N}{4==Me}
\end{center}
```

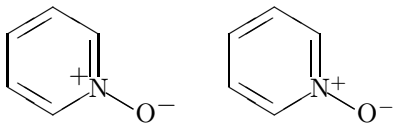
These codes generate the following structures:



The next examples illustrate two ways of drawing pyridine-N-oxide:

```
\sixheterovi[r{2+}]%
 {2==N}{2==O$^{\displaystyle -}$}
\sixheterovi{2==N$^{+}$}{2==O$^{-}$}
```

which yield the following results:

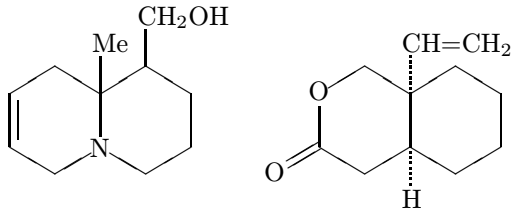


Note that pyridine nuclei with a ring nitrogen at a position other than the top or the bottom cannot be drawn by using such a specific command as `\pyridinev`.

For drawing fused heterocycles with two 6-membered rings, the `\decaheterov` command and related ones are designed to be capable of specifying any hetero-atoms in the nuclei:

```
\decaheterov[g]{9==N}{%
 1B==CH$_2$OH;{{10}B}==Me}
\decaheterov[]{7==O}{6D==O;9A==H;%
 {{10}A}==CH=CH$_2$}
```

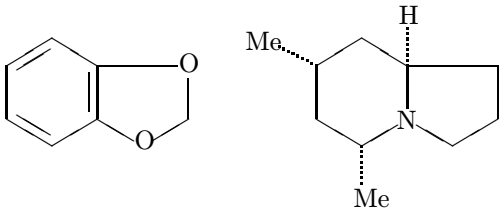
yields



The `\nonaheterov` command and related ones are also able to specify any hetero-atoms in the nuclei of fused heterocycles. For example:

```
\nonaheterov[egj]{1==O;3==O}{}
\nonaheterov[]{9==N}{5A==Me;7A==Me;8A==H}
```

produces



6 Further Techniques

More complicated structural formulas can be constructed by combining two or more structures created by XeTeX commands. These structures are combined within an outer `picture` environment, since XeTeX is based on the LaTeX `picture` environment and two or more `picture` environments can be nested. The technique is discussed in Chapter 14 of the on-line manual for XeTeX [8].

Another technique for constructing complicated formulas is to use a XeTeX command inside the

Table 2: Original XƎMTEX Files in NIFTY-Serve

no.	size (bytes)	data name
204	76093	`xymtexi.lzh`—XƎMTEX. An introduction (in Japanese)
202	77281	`xymtexj.lzh`—XƎMTEX by Example (in Japanese)
201	299053	`xymtex.lzh`—XƎMTEX for drawing chem. structures

argument of another XƎMTEX command. This technique is discussed in Chapter 15 of the same on-line manual cited above.

The book which was the original cause of all this work on TEX for chemistry [1] contains several commands for the use of chemical fields, e.g., counters for compounds and derivatives, various reaction arrows, `parbox`-like boxes for structural formulas, and chemical equation environments. These commands combined with the XƎMTEX ones are useful for drawing reaction schemes of multistep syntheses. Many illustrative examples are described in the Japanese edition of the XƎMTEX on-line manual. [9]

7 Program Availability

The original location supported by the author is the NIFTY-Serve archives (FPRINT library No. 7), from which you can take the compressed packages shown in Table 2. The XƎMTEX files as listed in Table 1 (including the reference manual of about 120 pages [8]) are also available on CTAN:[2]

```
tex-archive/macros/
latex209/contrib/xymtex
```

The present article has been typeset by using XƎMTEX within LATEX, where the top declaration of the document file is as follows:

```
\documentstyle[epic,carom,hetarom]%
{ltugboat}
```

References

[1] S. Fujita, *LATEX for Chemists and Biochemists. A Guide for Preparing Papers with Personal Computers* [in Japanese], Tokyo Kagaku Dozin, Tokyo (1993).

[2] R.T. Haas and K.C. O'Kane, "Typesetting chemical structure formulas with the text formatter TEX/LATEX", *Computers and Chemistry*, Vol. 11, No. 4, pp. 251–271 (1987).

[3] S. Podar, "Enhancements to the picture environment of LATEX", On-line manual for Version 1.2 dated July 14, 1986. Manual for Version 1.2 dated July 14, 1986.

[4] M.J. Wichura, *The PICTEX Manual*, TEX Users Group, Providence (1992).

[5] A.C. Norris and A.L. Oakley, "Electronic publishing and chemical text processing", in M. Clark, ed., *TEX. Applications, uses, methods*, Ellis Horwood, Chichester (1990), pp. 207–225; see also M. Ramek, "Chemical structure formulae and x/y diagrams with TEX", pp. 227–258.

[6] C. Kwok, "EEPIC. Extensions to epic and LATEX picture environment", On-line manual dated August 14, 1988.

[7] S. Fujita, "Typesetting structural formulae with the text formatter TEX/LATEX", *Computers and Chemistry*, Vol. 18, No. 2, pp. 109–116 (1994).

[8] S. Fujita, "XƎMTEX. A macro package for typesetting chemical structural formulas", On-line manual for XƎMTEX for Version 1.00 (1993). [Available from CTAN, see article.]

[9] S. Fujita, "XƎMTEX by example" [in Japanese], On-line manual for XƎMTEX (1994). [Available from NIFTY-Serve, see Table 2.]

◇ Shinsaku Fujita
Ashigara Research Laboratories
Fuji Photo Film Co., Ltd.
Minami-Ashigara, Kanagawa-ken, 250-01 Japan
Email: `hbh00445@niftyserve.or.jp`

[2] Thanks to the kind volunteer efforts of Ms. M. Burbank and Ms. H. Ase.

Calendar

1995

Apr 4 — UK TEX Users' Group, University of Warwick. BIBTEX and *MakeIndex* tutorial. For information, e-mail `uktug-enquiries@ftp.tex.ac.uk`

May 24 — NTG 15th Meeting, Universiteit Twente, Enschede, The Netherlands. For information, contact Gerard van Nes (`vannes@ecn.nl`).

Apr 29 – May 1 — BachoTEX '95, Poland. For information, contact Hanna Kołodzeijska, (`gust@camk.edu.pl`).

Jun 1 – 2 — GUTenberg '95, "Graphique, TEX et PostScript", La Grande Motte, France. For information, call (33-1) 30-87-06-25, or e-mail `treasorerie.gutenberg@ens.fr` or `aro@lirmm.fr`.

Jun 1 – 2 — IWHD '95: International Workshop on Hypermedia Design, Montpellier, France. For information, contact the conference secretariat, Corine Zicler, LIRMM, Montpellier ((33) 6741 8503, `zicler@lirmm.fr`).

Jul 6 — DANTE TEX–Stammtisch at the Universität Bremen, Germany. For information, contact Martin Schröder (`l15d@zfn.uni-bremen.de`; telephone 0421/628813). 18:30, Universität Bremen MZH, 4th floor, across from the elevator.

Jul 24 – 28 — **TUG 16th Annual Meeting**: Real World TEX, St. Petersburg Beach, Florida. For information, send e-mail to `tug95c@scri.fsu.edu`. (For a preliminary announcement, see *TUGboat* 15, no. 2, p. 160.)

Jul 27 — TEX and Semitic Languages, Technion, Haifa, Israel. (For information, contact one of the organizers: Dan Berry `dberry@cs.technion.ac.il` or Yannis Haralambous `Yannis.Haralambous@univ-lille1.fr`.)

Aug 10 — DANTE TEX–Stammtisch at the Universität Bremen, Germany. (For details, see Jul 6.)

Sep 4 – 8 — EuroTEX '95, Papendal, Arnhem, The Netherlands. For information, contact `eurotex@cs.ruu.nl`.

Sep 7 — DANTE TEX–Stammtisch at the Universität Bremen, Germany. (For details, see Jul 6.)

Sep 14 – 15 — DANTE e.V., 13th general meeting, Humboldt-Universität, Berlin, Germany. (For information, contact Christiane Schöbel `dante-mv13@rz.hu-berlin.de`.)

Sep 14 — DANTE TEX–Stammtisch, Wuppertal, Germany. For information, contact Andreas Schrell (`Andreas_Schrell@FernUni-Hagen.DE`, telephone 0202/502354). Second Thursday, 19:30, Gaststätte Yol, Ernststraße 43, (near Robert-Daum-Platz), 42117 Wuppertal.

Oct 2 – 5 — CyrTUG'95 Annual Meeting, Protvino (Moscow region), Russia. (For information, contact `cyrtug@mir.msk.su`.)

TUG Courses, San Francisco, California
(For information, contact `tug@tug.org`.)

Oct 9 – 13 — Beginning/Intermediate TEX

Oct 16 – 20 — Intensive LATEX

Oct 23 – 27 — Modifying LATEX Document Classes

Oct 30 – Nov 3 — Advanced TEX and Macro Writing

Oct 10 — DANTE TEX–Stammtisch at the Universität Bremen, Germany. For information, contact Martin Schröder (`MS@Dream.HB.North.de`; telephone 0421/628813). First Tuesday (if not a holiday), 18:00, Universität Bremen MZH, 28359 Bremen, 4th floor, across from the elevator.

Oct 12 — DANTE TEX–Stammtisch, Wuppertal, Germany. (For details, see Sep 14.)

Status as of 31 December 1994

Nov 7 DANTE TEX–Stammtisch at the Universität Bremen, Germany. (For details, see Oct 10.)

Nov 9 DANTE TEX–Stammtisch, Wuppertal, Germany. (For details, see Sep 14.)

Dec 5 DANTE TEX–Stammtisch at the Universität Bremen, Germany. (For details, see Oct 10.)

Dec 14 DANTE TEX–Stammtisch, Wuppertal, Germany. (For details, see Sep 14.)

1996

Jan 9 DANTE TEX–Stammtisch at the Universität Bremen, Germany. (For details, see Oct 10.)

Feb 6 DANTE TEX–Stammtisch at the Universität Bremen, Germany. (For details, see Oct 10.)

Mar 5 DANTE TEX–Stammtisch at the Universität Bremen, Germany. (For details, see Oct 10.)

Mar 27 – 29 DANTE '96 and 14th general meeting of DANTE e.V., Universität Augsburg, Germany. For information, contact Gerhard Wilhelms (dante96@Uni-Augsburg.de).

Apr 2 DANTE TEX–Stammtisch at the Universität Bremen, Germany. (For details, see Oct 10.)

Apr 7 – 10 EP96, the International Conference on Electronic Documents, Document Manipulation and Document Dissemination, Xerox Palo Alto Research Center, Palo Alto, California. *Deadline for submission of papers: 4 December 1995.* For information, contact ep96@xsoft.xerox.com.

May 7 DANTE TEX–Stammtisch at the Universität Bremen, Germany. (For details, see Oct 10.)

Jul 18 – 21 SHARP 1996: Society for the History of Authorship, Reading and Publishing, Fourth Annual Conference, Worcester, Massachusetts. *Deadline for proposals: 20 November 1995.* For information, contact the American Antiquarian Society, cfs@mark.mwa.org.

For additional information on the events listed above, contact the TUG office (415-982-8449, fax: 415-982-8559, e-mail: tug@tug.org) unless otherwise noted.

Late-Breaking News

Production notes

Mimi Burbank

I oversaw the production of this issue of *TUGboat* and I had to manage a production team working in three different time zones and spanning two continents. But with e-mail and ftp and so forth, time and distance were not a problem. It soon was obvious that some of the production team were at work for practically all of the 24-hour day. Each member was involved in the successful completion of this issue, as well as maintaining and upgrading the system used at SCRI. Bandwidth was often a problem for those "across the pond" and Mimi's main activity was running and previewing files and reporting back to those across the ocean about layout.

Electronic input for articles in this issue was received by e-mail as well as retrieved from remote sites by anonymous ftp. In addition to text, the input to this issue includes METAFONT source code, 38 `.fd` files, and 11 `.vf` files. There were a considerable number and variety of PostScript files. One article contained 39 figures, and required 81 files to produce final output. Over 200 files were used (as input files) to generate final copy; over 300 files represent fonts (`.tfm` and rasters), device-specific translations, earlier versions of files, auxiliary information, and records of correspondence with authors and referees. The Y&Y advertisement was received via anonymous ftp as a PostScript file.

All articles were received as fully tagged for *TUGboat*, using either `plain`-based or LaTeX conventions described in the Authors' Guide (see *TUGboat* 10, no. 3, pages 378 – 385). 80% of the articles received were in LaTeX 2ε. Several authors requested copies of the current version of LaTeX 2ε macros for *TUGboat*, and we were happy to provide these.

Font work was required on all of the articles in the "Font Forum" section. The article by Jeffrey (page 79) used metrics for Adobe Times which were generated in 1994. Unfortunately, a major change to the fontinst macros took place in mid-1995, resulting in different stretch and shrink values in all the PostScript font metrics distributed as PSNFSS. Since Alan's article deals explicitly with the effects of changing TeX's parameters relating to setting text, using the current Adobe Times PSNFSS metrics caused disastrous results, so we had to maintain a copy of the old metrics for this paper.

The production team has been experimenting with a pre-release of changes to dvips that allow automatic partial-downloading of Type1 fonts. The much smaller PostScript files produced are very convenient when they have to be transferred across a slow transatlantic ftp link. The changes to dvips were made by Sergey Lesenko, and are described in a paper which will appear in the 1995 proceedings issue. We hope that they will appear in standard dvips soon. Type1 versions of the CM fonts are now used as standard to avoid printing complications on different devices.

Output

Though individual articles were worked on by members of the production team on their local computer systems, the final output was prepared at SCRI on an IBM RS6000 running AIX, using the *Web2C* implementation of TeX. Output was printed on a QMS 680 print system at 600 dpi.

Future Issues

The next issue will be a theme issue and will be guest-edited by Malcolm Clark. 16(3) will be the TUG'95 proceedings issue, and we plan for 16(4) to be a bilingual issue featuring articles in both Russian and English.

Topics for future theme issues will be announced as plans become firm. Suggestions are welcome for prospective topics and guest editors. Send them to the Editor, Barbara Beeton (see address on page 3), or via electronic mail to `TUGboat@ams.org`.

Coming Next Issue

Guest-edited issue

The next issue of *TUGboat*, guest-edited by Malcolm Clark, focuses on 'portable' electronic documents. It contains articles on the Standard Generalized Markup Language, bringing in its relationship to HTML (Hypertext Markup Language) and the World Wide Web. The other strands are Adobe's Portable Document Format (a hypertext-capable version of PostScript, and more), which can be generated from existing TeX and LaTeX documents, and packages which may be included with LaTeX to produce hypertexts suitable for reading at a screen, rather than paper. The brave new world it heralds is one where the tyranny of paper is broken, and all 'documents' are truly virtual. Xanadu looms through the mists!

- *A Practical Introduction to SGML*
  Michel Goossens and Janne Saarela
- *From LaTeX to HTML and Back*
  Michel Goossens and Janne Saarela
- *The Inside Story of Life at Wiley with SGML, LaTeX and Acrobat*
  Geeti Granger
- *LaTeX, HTML and PDF,* or *the entry of TeX into the world of hypertext*
  Yannis Haralambous and Sebastian Rahtz
- *HTML & TeX: Making them sweat*
  Peter Flynn
- *The Hyperlatex Story*
  Otfried Schwarzkopf
- *The Los Alamos E-print Archives: HyperTeX in Action*
  Mark D. Doyle

Institutional Members

The Aerospace Corporation,
El Segundo, *California*

* Air Force Institute of Technology,
Wright-Patterson AFB, *Ohio*

American Mathematical Society,
Providence, *Rhode Island*

* ArborText, Inc.,
Ann Arbor, *Michigan*

* Brookhaven National Laboratory,
Upton, *New York*
Pasadena, California¡

CNRS - IDRIS,
Orsay, *France*

CERN, *Geneva*, *Switzerland*

* College Militaire Royal de Saint Jean, *St. Jean*, *Quebec*, *Canada*

College of William & Mary,
Department of Computer Science,
Williamsburg, *Virginia*

Communications
Security Establishment,
Department of National Defence,
Ottawa, *Ontario*, *Canada*

*CS*TUG, *Praha*, *Czech Republic*

Elsevier Science Publishers B.V.,
Amsterdam, *The Netherlands*

* Fermi National Accelerator Laboratory, *Batavia*, *Illinois*

Florida State University,
Supercomputer Computations Research, *Tallahassee*, *Florida*

Grinnell College,
Noyce Computer Center,
Grinnell, *Iowa*

Hong Kong University of
Science and Technology,
Department of Computer Science,
Hong Kong

Institute for Advanced Study,
Princeton, *New Jersey*

Institute for Defense Analyses,
Communications Research
Division, *Princeton*, *New Jersey*

Iowa State University,
Ames, *Iowa*

Los Alamos National Laboratory,
University of California,
Los Alamos, *New Mexico*

Marquette University,
Department of Mathematics,
Statistics and Computer Science,
Milwaukee, *Wisconsin*

Mathematical Reviews,
American Mathematical Society,
Ann Arbor, *Michigan*

* Max Planck Institut
für Mathematik,
Bonn, *Germany*

New York University,
Academic Computing Facility,
New York, *New York*

Nippon Telegraph &
Telephone Corporation,
Basic Research Laboratories,
Tokyo, *Japan*

* Personal TEX, Incorporated,
Mill Valley, *California*

Princeton University,
Princeton, *New Jersey*

Smithsonian Astrophysical
Observatory, *Cambridge*,
Massachusetts

Space Telescope Science Institute,
Baltimore, *Maryland*

Springer-Verlag,
Heidelberg, *Germany*

* Stanford Linear Accelerator
Center (SLAC),
Stanford, *California*

Stanford University,
Computer Science Department,
Stanford, *California*

Texas A & M University,
Department of Computer Science,
College Station, *Texas*

* United States Naval
Postgraduate School,
Monterey, *California*

United States Naval Observatory,
Washington DC

University of California, Berkeley,
Center for EUV Astrophysics,
Berkeley, *California*

University of California, Irvine,
Information & Computer Science,
Irvine, *California*

University of Canterbury,
Christchurch, *New Zealand*

University College,
Cork, *Ireland*

University of Delaware,
Newark, *Delaware*

University of Groningen,
Groningen, *The Netherlands*

Universität Koblenz–Landau,
Koblenz, *Germany*

University of Manitoba,
Winnipeg, *Manitoba*

University of Oslo,
Institute of Informatics,
Blindern, *Oslo*, *Norway*

* University of Southern California,
Information Sciences Institute,
Marina del Rey, *California*

University of Stockholm,
Department of Mathematics,
Stockholm, *Sweden*

University of Texas at Austin,
Austin, *Texas*

Università degli Studi di Trieste,
Trieste, *Italy*

Uppsala University,
Uppsala, *Sweden*

Vrije Universiteit,
Amsterdam, *The Netherlands*

Wolters Kluwer,
Dordrecht, *The Netherlands*

Yale University,
Department of Computer Science,
New Haven, *Connecticut*

(52 institutions listed)

TEX Consulting & Production Services

Information about these services can be obtained from:

TEX Users Group
1850 Union Street, #1637
San Francisco, CA 94123, U.S.A.
Phone: +1 415 982-8449
Fax: +1 415 982-8559
Email: tug@tug.org

North America

Anagnostopoulos, Paul C.
Windfall Software,
433 Rutland Street, Carlisle, MA 01741;
(508) 371-2316; greek@windfall.com
We have been typesetting and composing high-quality books and technical Publications since 1989. Most of the books are produced with our own public-domain macro package, ZzTEX, but we consult on all aspects of TEX and book production. We can convert almost any electronic manuscript to TEX. We also develop book and electronic publishing software for DOS and Windows. I am a computer analyst with a Computer Science degree.

Cowan, Dr. Ray F.
141 Del Medio Ave. #134, Mountain View, CA 94040;
(415) 949-4911; rfc@netcom.com
Twelve Years of TEX and Related Software Consulting: Books, Documentation, Journals, and Newsletters
TEX & LATEX macropackages, graphics; PostScript language applications; device drivers; fonts; systems.

Hoenig, Alan
17 Bay Avenue, Huntington, NY 11743; (516) 385-0736
TEX typesetting services including complete book production; macro writing; individual and group TEX instruction.

NAR Associates
817 Holly Drive E. Rt. 10, Annapolis, MD 21401;
(410) 757-5724
Extensive long term experience in TEX book publishing with major publishers, working with authors or publishers to turn electronic copy into attractive books. We offer complete free lance production services, including design, copy editing, art sizing and layout, typesetting and repro production. We specialize in engineering, science, computers, computer graphics, aviation and medicine.

Ogawa, Arthur
40453 Cherokee Oaks Drive,
Three Rivers, CA 93271-9743;
(209) 561-4585
Experienced in book production, macro packages, programming, and consultation. Complete book production from computer-readable copy to camera-ready copy.

Quixote Digital Typography, Don Hosek
555 Guilford, Claremont, CA 91711;
(909) 621-1291; Fax: (909) 625-1342;
dhosek@quixote.com
Complete line of TEX, LATEX, and METAFONT services including custom LATEX style files, complete book production from manuscript to camera-ready copy; custom font and logo design; installation of customized TEX environments; phone consulting service; database applications and more. Call for a free estimate.

Richert, Norman
1614 Loch Lake Drive, El Lago, TX 77586;
(713) 326-2583
TEX macro consulting.

Type 2000
16 Madrona Avenue, Mill Valley, CA 94941;
(415) 388-8873; Fax: (415) 388-8865
pti@crl.com
$2.50 per page for 2000 DPI TEX and PostScript camera ready output! We provide high quality and fast turnaround to dozens of publishers, journals, authors and consultants who use TEX. Computer Modern, PostScript and METAFONT fonts available. We accept DVI and PostScript files only and output on RC paper. $2.25 per page for 100+ pages, $2.00 per page for 500+ pages; add $.50 per page for PostScript.

Outside North America

TypoTEX Ltd.
Electronical Publishing, Battyány u. 14. Budapest,
Hungary H-1015; (036) 11152 337
Editing and typesetting technical journals and books with TEX from manuscript to camera ready copy. Macro writing, font designing, TEX consulting and teaching.