TeXConf 2019（2019 年 10 月 12 日）

原ノ味フォントと ToUnicode CMap

細田 真道
http://www.trueroad.jp

概要

原ノ味明朝・原ノ味角ゴシック（原ノ味フォント）は、Adobe-Identity-0 (AI0) フォントである源ノ明朝・源ノ角ゴシック（源ノフォント）を、Adobe-Japan1 (AJ1) フォントになるよう組み替えたフォントである。AI0 フォントは AJ1 フォントと同様の使い方ができず、使用が困難なケースがある。そこで源ノフォントを AJ1 へ組み替えることにより、豊富なウェイト数を有する源ノフォントのデザインをそのままに、従来の AJ1 フォントと同様の使い方ができるようにした。本稿では、こうした原ノ味フォントの特徴を紹介する。そのため、まず AJ1 と AI0 を比較し、源ノフォントが一部の TEX エンジン・DVI ドライバや Ghostscript で使用困難となる理由を述べる。そして、それを解決するために作成した、源ノフォントを AJ1 へ組み替える原ノ味フォント生成プログラムと、原ノ味フォントの使用例を紹介する。次に、PDF からテキスト抽出する際に使われる ToUnicode CMap について述べ、源ノフォントを使うと一部のワークフローではテキスト抽出時に文字化けする PDF が生成されてしまう場合があることを示す。そして、すべてのテキストが抽出できる必要がある PDF/A-2u 規格について述べ、規格上は AJ1 であれば ToUnicode CMap が無くてもよいこと、それでも正しくテキスト抽出できることを示す。最後に LuaTEX は AJ1 でも ToUnicode CMap を生成し、一部のグリフで正しくないテキストが抽出される場合があることを示す。そこで PDF/A-2u 規格を満たしたまま、PDF から原ノ味フォントの ToUnicode CMap を削除し、正しいテキスト抽出ができるようにするツールを作成したので紹介する。

1 はじめに

源ノ明朝 [1]・源ノ角ゴシック [2]（以下、源ノフォント）は SIL Open Font License 1.1 [3] に基づくオープンソースの Pan-CJK フォントである。日本語だけでなく CJK 各言語で使われる多数のグリフを収録し、明朝・ゴシックそれぞれ 7 ウェイトという豊富なウェイト数のフォントが配布されている。フォントの形式は CID-keyed OpenType/CFF であるが、従来の同形式の日本語フォントは、文字コレクションとして Adobe-Japan1（以下、AJ1）[4] が使われていたのに対して、源ノフォントは Adobe-Identity-0（以下、AI0）が使われている。このため、日本語フォントが AJ1 であることを前提としたシステムで、AI0 である源ノフォントを使用するのは困難である。

各種 TEX エンジン・DVI ドライバにおいては、関係各位のご尽力により AI0 対応・源ノフォント対応が進められてきた。その結果、最新の TEX Live 2019 までに、モダンな LuaTEX や XETEX では他の日本語フォントとほぼ同様に使用でき、upTEX でも DVI ドライバが dvipdfmx であれば、一部を除きほぼ同様に使用できるようになっている。pTEX においても DVI ドライバに dvipdfmx を用いて PXufont パッケージ [5] を利用することで源ノフォントを使用できる。しかし pTEX しか選択できない（モダンな LuaTEX や XETEX に移行できず、upTEX も使えない）場合で、PXufont パッケージも使えない場合には、源ノフォントを使用することは困難である。また、upTEX が選択できる場合

や、pTEX で PXufont パッケージを使ってよい場合であっても、DVI ドライバに dvipdfmx ではなく dvips を利用したい場合には、源ノフォントを使用することは困難である。

Ghostscript は最初から源ノフォントを使うことを前提にして作成された（例えば LilyPond [6] で源ノフォントを使って生成した）PostScript ファイルであれば、問題なく画面表示や PDF 変換などが可能である。しかし、AJ1 フォントを前提とした（例えば dvips で Ryumin-Light や GothicBBB-Medium を前提に生成した）PostScript ファイルを、源ノフォントで代替して画面表示や PDF 変換などすることは困難である。

さらに、源ノフォントを使った PDF は、正しい ToUnicode CMap が埋め込まれていないと、PDF からのテキスト抽出[1)]が文字化けする。PDF を生成するワークフローによっては、ToUnicode CMap が誤っていたり生成できなかったりして、表示や印刷では問題なくてもテキスト抽出時に文字化けしてしまうことがある。

一方、原ノ味フォント [7] は AJ1 なので、豊富なウェイト数を有する源ノフォントのデザインをそのままに、従来の AJ1 である日本語フォントと同様の使い方ができる。また、PDF に ToUnicode CMap が無くてもテキスト抽出が文字化けしない。

本稿では、こうした原ノ味フォントの特徴を紹介する。そのため、まず AJ1 と AI0 を比較し、源ノフォントが一部の TEX エンジン・DVI ドライバや Ghostscript で使用困難となる理由を述べる。そして、それを解決するために作成した、源ノフォントを AJ1 へ組み替える原ノ味フォント生成プログラム [8] と、原ノ味フォントを各種 TEX エンジン・DVI ドライバや Ghostscript で使用する例を紹介する。次に、PDF からテキスト抽出する際に使われる ToUnicode CMap について述べ、源ノフォントを使うと一部のワークフローでは間違った ToUnicode CMap が PDF に埋め込まれたり、あるいは ToUnicode CMap が生成できなかったりして、表示や印刷では問題なくてもテキスト抽出時に文字化けが発生する PDF が生成されてしまう場合があることを示す。そして、すべてのテキストが抽出できる必要がある PDF/A-2u 規格について述べ、規格上は AJ1 であれば ToUnicode CMap が無くてもよいことになっており、それでも正しくテキスト抽出できることを示す。最後に、LuaTEX は AJ1 フォントを使っても ToUnicode CMap を生成して AI0 として PDF へ埋め込む動作を行うことと、この動作の影響で縦書きなど一部のグリフで正しくないテキスト抽出がされる場合があることを示す。そこで PDF/A-2u 規格を満たしたまま、PDF から原ノ味フォントの ToUnicode CMap を削除して AJ1 に戻し、正しいテキスト抽出ができるようにするツールを作成 [9] したので紹介する。

2 AJ1 と AI0

2.1 フォントと CID

フォントは、いくつかの文字について、統一されたデザインのグリフ（字形）を収録したものであり、OpenType など様々な形式がある。源ノフォント [1][2] は CID-keyed OpenType/CFF 形式の Pan-CJK フォントである。Pan-CJK なので、CJK 各言語で使用される多数の文字を収録しており、収録している各文字について、グリフとして形状のアウトラインデータなどを持っている。

CID (Character ID) は様々な種類の文字 (Character) ひとつひとつに付けられたユニークな識別子 (ID) で、0 から始まる番号である[2)3)]。CID-keyed は「CID キー方式」とも呼ばれ、CID をキーとして使いたい文字のグリフを指定する方式であることを示している。つまり、CID-keyed フォントを

1） PDF viewer からのコピー&ペーストなど。

2） 0 番は CID+0、1 番は CID+1 のように、“CID+” の後に ID を 10 進数の数字で付けて表記する。

3） OpenType 登場以前の古いフォント形式に CID フォントというものもあるが、本稿では触れない。

表 1　文字と CID の対応例

	Unicode	AJ1	源ノ明朝 1.001	源ノ角ゴシック 2.001
あ	U+3042	CID+843	CID+1461	CID+1461
漢	U+6F22	CID+1533	CID+24743	CID+24227
☃	U+2603	CID+8218	CID+1274	CID+1281
𫝆	U+32FF	CID+23058	—	CID+2184
𠮟	〃	CID+23059	—	CID+65359
飴	U+98F4	CID+7634	CID+45263	CID+44358
飴	〃	CID+1151	CID+61214	CID+62049
見	U+898B	CID+1887	CID+38198	CID+37348
見	U+2F92	〃	〃	〃

用いるアプリケーションは、使いたい文字を CID 番号で指定し、そのグリフのアウトラインを取得して、表示や印字などをする[4)]。表 1 に、いくつかの文字について対応する CID を示す。

通常、アプリケーションは CID ではなく JIS X 0208 や Unicode などの文字コード規格で文字を取り扱っているものが多い。JIS X 0208 は日本語でよく使われる文字を集め、それらを識別する番号を付与したものである[5)]。Unicode は JIS X 0208 も含めた世界中の様々な規格にある文字などを集め、それらを識別する番号を付与したものである[6)]。これらは CID とは全く異なった体系にあるため、変換テーブル（マッピング）を用意して CID に変換する必要がある。

CID-keyed フォントは、どのような文字コレクションか（どのような文字を収録しているか）を示す ROS という情報を持っている。ROS は “Registry”（登録者）、“Ordering”（版）、“Supplement”（追補）の頭文字で、この 3 つをハイフンでつないだ名前で呼ばれる。Registry は Ordering 発行者の名前、Ordering は文字コレクションを識別する名前である。Supplement は 0 から始まる番号であり、新版が制定されるときに 1 増加する。例えば、AJ1 の最新版である Adobe-Japan1-7 (AJ1-7) は、Registry が “Adobe”、Ordering が “Japan1”、Supplement が “7” であることを示している。同様に Adobe-Identity-0 (AI0) は、Registry が “Adobe”、Odering が “Identity”、Supplement が “0” であることを示している。AJ1 と AI0 では、CID やマッピングの扱いに異なるところがあるため、それぞれについて説明する。

4） アプリケーション自身は直接フォントを扱わず、ライブラリや API が処理しているケースも多いと思われる。

5） 区と点それぞれ 10 進数で表し「16 区 1 点」や、それぞれ 2 桁の 10 進数をハイフンでつなぎ “16-01” などと表記する。これを符号化（エンコーディング）するには ISO-2022-JP・Shift_JIS・EUC-JP などを用いる。

6） “U+” の後に 16 進数 4〜6 桁を付けて表記する。これを符号化するには UTF-8・UTF-16・UTF-32 などを用いる。

表 2　Adobe-Japan1 (AJ1)

	制定年	追加された CID			追加数	合計
AJ1-0	1992	0	〜	8283	8,284	8,284
AJ1-1	1993	8284	〜	8358	75	8,359
AJ1-2	1993	8359	〜	8719	361	8,720
AJ1-3	1998	8720	〜	9353	634	9,354
AJ1-4	2000	9354	〜	15443	6,090	15,444
AJ1-5	2002	15444	〜	20316	4,873	20,317
AJ1-6	2004	20317	〜	23057	2,741	23,058
AJ1-7	2019	23058	〜	23059	2	23,060

2.2　Adobe-Japan1 (AJ1)

Adobe-Japan1 (AJ1) [4] は日本語で使われる様々な文字を集めた文字コレクションで、それらを識別するため、ひとつひとつの文字に CID を割り当てたものである。

2.2.1　制定

表 2 に AJ1 の制定について示す。1992 年に最初の Adobe-Japan1-0 (AJ1-0) が制定され、CID+0 から CID+8283 までの 8,284 個について、どの文字と対応するのか定義された。その後は順次 CID を追加した Supplement が制定されていき、2019 年制定の Adobe-Japan1-7 (AJ1-7) では CID+23058 と CID+23059 の 2 個が追加された。各 Supplement には、それ以前の Supplement がすべて含まれているため、AJ1-7 には AJ1-0 から AJ1-6 までの定義も収録されている。よって、最新の AJ1-7 では CID+0 から CID+23059 までの 23,060 個について、どの文字と対応するか定義されている。Supplement を区別する必要のあるときは Adobe-Japan1-7 や AJ1-7

表 3　文字幅 (AJ1)

文字幅	AJ1 CID の範囲
プロポーショナル	1 〜 230
	9354 〜 9737
	15449 〜 15975
	20317 〜 20426
半角	231 〜 632
	8718 〜 8719
	12063 〜 12087
三分	9758 〜 9778
四分	9738 〜 9757
全角	その他

のように表記するが、区別する必要のないときは Adobe-Japan1 や AJ1 のように表記する。

2.2.2　特徴

CID-keyed の日本語フォントには、文字コレクションに AJ1 を用いたものが数多くある[7)]。AJ1 フォントであれば、どのフォントであっても同じ CID は同じ文字である。例えば、表 1 のように AJ1 の CID+843 は「あ」、CID+1533 は「漢」、CID+8218 は「⛄」などと決まっている。また、表 3 に示す通り、CID ごとに、文字幅がプロポーショナル、半角、三分、四分、全角のいずれかに決まっている。

また、AJ1 は Unicode などの、通常の文字コード規格では区別しない文字を区別することがある[8)]。例えば、表 1 にある「令和」の組み文字は、AJ1 では CID+23058 が横書き用の「㍿」、CID+23059 が縦書き用の「𠀋」、というように 2 種類あるのに対して、Unicode では区別されておらず両方とも U+32FF である。同じく「飴」は、AJ1 では CID+7634 が JIS2004 字形[9)]の「飴」、CID+1151 が JIS90 字形[10)]の「飴」、というように 2 種類あるのに対して、Unicode では区別されておらず両方とも U+98F4 である。逆に Unicode では区別されているが AJ1 では区別されないものもある[11)]。例えば、表 1 にある「見」は、Unicode では U+898B と U+2F92 の 2 種類があるが、AJ1 ではいずれも CID+1887 にマッピングされる[12)]。

AJ1 は文字とユニークな ID のペアを集めたものなので、Unicode などと同じ文字コードの一種とも言える。しかし、Unicode などとは文字を区別するしないの基準をはじめ、体系が全く異なる。アプリケーションが Unicode などから AJ1 へ変換するには、CMap リソースを使う方法と、OpenType の cmap テーブルを使う方法がある[13)]。

2.2.3　CMap リソース

pTEX や upTEX が出力した DVI ファイルを、dvipdfmx で AJ1 フォントを使って処理する場合に使用される方法である。また、dvips [14)]で AJ1 フォントを使うことを前提に出力した PostScript ファイルを、PostScript インタプリタ[15)]で処理する際に使用される方法でもある。

CMap リソース [10] は、Unicode などから、AJ1 のような文字コレクションの CID へのマッピングが示されたテーブルである。AJ1 用（変換先が AJ1）のものは、AJ1 のディレクトリ[16)]の下にある `CMap` ディレクトリ内に格納されている。

入力（変換元）の文字コードやエンコーディング、出力（変換先）で使用したい JIS90 字形・JIS2004 字形や横書き・縦書き別に、ファイルが用意されている。これにより、使用する CMap リソースを切り替えることで、入力の文字コード・エンコーディングを選択できるだけではなく、Unicode などでは区別しない JIS90 字形・JIS2004 字形、横書き・縦書きの選択を行うことができる。

7）源ノフォントは、日本語用含め、すべて非 AJ1 である。また、日本語フォントでも非 AJ1 のものは存在する。
8）AJ1 は見た目が異なるものを区別するようである。Unicode などは「デザインの差」の範囲は区別しない。
9）JIS X 0213:2004 の例示字形に準じたもの。
10）JIS X 0208-1990 の例示字形に準じたもの。
11）基本的に Unicode は、由来などが異なれば見た目が同じでも区別するが、AJ1 は区別しない。
12）公式の “Adobe-Japan1-UCS2” [11] で CID+1887 から Unicode へ逆変換する際には U+898B になる。
13）OpenType 登場以前の古い CID フォントは cmap テーブルが無いので、CMap リソースしか使えない。
14）かつて広く使われたが、現在は dvipdfmx の方が主流。
15）Ghostscript など。
16）現在は `Adobe-Japan1-7` ディレクトリ。

例えば素の（OTF パッケージを使用しない）pTEX は、和文文字として DVI ファイルに JIS コード[17]を出力するため、使用すべき CMap リソースは、横書き用 “H” と縦書き用 “V” である。これらは JIS90 字形の CID を出力するが、JIS2004 字形にしたい場合は、別途配布 [12] されている横書き用 “2004-H” と縦書き用 “2004-V” を使えばよい。一方、素の upTEX は、和文文字として DVI ファイルに Unicode [18]を出力するため、使用すべき CMap リソースは、JIS90 字形ならば横書き用 “UniJIS-UTF16-H” と縦書き用 “UniJIS-UTF16-V”、JIS2004 字形ならば横書き用 “UniJIS2004-UTF16-H” と縦書き用 “UniJIS2004-UTF16-V” である[19]。これらの CMap リソースはいずれも TEX Live に含まれており、TEX Live で pTEX・upTEX が使えるならばすべて使えるはずである。

CMap リソースを使う方法は、条件に応じてフォントと CMap リソースの双方を指定する必要があり、アプリケーションの設定項目は煩雑になりがちである。

2.2.4 cmap テーブル

LuaTEX や XETEX で OpenType フォントを扱う場合に使用される方法である。また、LilyPond [6] では PDF 出力、PostScript 出力、EPS 出力のいずれでも、この方法が使用される。

cmap テーブルは OpenType フォントのファイル内に含まれているテーブルで、CID-keyed の場合は文字コードから CID へのマッピングを示したものである。CMap リソースとほぼ同じ役割だが、異なる点として、CMap リソースはフォントファイルとは独立した別のファイルであるのに対して、cmap テーブルはフォントファイルに内蔵したものであることが挙げられる。また、CMap リソースは複数のファイルを切り替えることで、変換元の文字コードやエンコーディングを選択したり、変換先の JIS90 字形・JIS2004 字形や横書き・縦書きを選択することができるが、cmap テーブルでは基本的に変換元は Unicode のみ[20]、変換先も固定[21]で切り替えることはできず、字形や縦横の選択には OpenType feature のような別の仕組みを使う[22]。LuaTEX や XETEX はもちろん、LilyPond もこの OpenType feature に対応しており、字形などの切り替えが可能である。

cmap テーブルを使う方法は、CMap リソースを使う方法とは異なり、フォントを指定するだけで完結し、字形などの切り替えも OpenType feature に統一されているため、アプリケーションの設定項目をシンプルにすることができる。

2.3 Adobe-Identity-0 (AI0)

Adobe-Identity-0 (AI0) は「特別な目的」に使われる文字コレクションで、何か特定の文字を集めたものではなく、文字と CID の組み合わせを規格として制定したものではない。

2.3.1 特徴

AJ1 は「日本語のため」の文字コレクションであるのに対して、AI0 は「特別な目的」の文字コレクションである、という違いがある。

AJ1 は「日本語のため」に、日本語の様々な文字を集めて CID を割り当てたものである。AJ1 フォントには、AJ1 で定義されていない文字（日本語では使われない文字）や、同じ文字の複数バリ

17） 文字コードに JIS X 0208 を、エンコーディングに ISO-2022-JP を使ったもの。

18） 文字コードに Unicode を、エンコーディングに UTF-16 (UTF-32) を使ったもの。

19） “UniJIS-UTF16-H” や “UniJIS2004-UTF16-H” は、upTEX で全角幅が前提となっているクォート 4 文字（U+2018「‘」U+2019「’」U+201C「“」U+201D「”」）をプロポーショナル幅のグリフにマッピングしている。そこで、これら 4 文字だけを VF で別の JFM（通常の JFM “*-h” に対してクォート用は “*-hq”） に割り当て、“*-hq” はこれらが全角幅にマッピングされる “UniJIS-UCS2-H” を使う設定にし、文字幅の問題発生を回避している。

20） 規格上は Unicode 以外の cmap テーブルも存在しているが、現在は Unicode が推奨され、ほとんど使われない。

21） JIS90 字形・JIS2004 字形どちらになるかはフォントによって異なる。また、必ず横書き用となる。

22） cmap テーブルで得たデフォルトの CID から、GSUB テーブルを用いて異なる字形や縦書きの CID を得る。

エーションを収録することはできない。AJ1 フォント同士ならば、別のフォントであっても、同じ CID は必ず同じ文字でなければならない。一方、AI0 は文字も CID の割り当ても文字幅も、何も決められておらず、フォント制作者が自分で設定した「特別な目的」に合わせて自由に決めることができる。つまり、AJ1 で定義されていない文字でも、同じ文字の複数バリエーションでも収録できるし、別のフォントでは、同じ CID が別の文字になって構わない。つまり、AI0 はフォント制作者が個々のフォントごとに決めた、独自の文字コレクションが使われていることを意味している。

源ノフォントは AI0 のフォントであり、表 1 に示すように、源ノ明朝と源ノ角ゴシックの間でも CID が異なる。現在はウェイトが違うだけならば同じ CID だが、今後は変更されるかもしれないし、バージョンが違えば CID が異なるかもしれない。また、AJ1 と同じ字（同じ Unicode）であっても、AJ1 の文字幅とは異なっているものがある。

アプリケーションが Unicode などから CID へ変換するには、AJ1 と同様、CMap リソースを使う方法と OpenType の cmap テーブルを使う方法がある。

2.3.2 CMap リソース

AI0 フォントに対して AJ1 の CMap リソースを使うと、ROS の不一致[23]でエラーとなる（DVI ドライバに dvipdfmx を使用した場合など）か、文字と CID の対応関係が異なるため、盛大に文字化けする（DVI ドライバに dvips を使用して出力した PostScript ファイルを、Ghostscript で処理した場合など）。そのため、フォントごとの CMap リソースが必要となる。

源ノフォントでは、表 4 の通り、CMap リソース

表 4　源ノフォントの日本語用 CMap リソース

源ノ明朝	UniSourceHanSerifJP-UTF32-H
	UniSourceHanSerifJP-UTF16-H
源ノ角ゴシック	UniSourceHanSansJP-UTF32-H
	UniSourceHanSansJP-UTF16-H

が配布されている[24)25)]が、変換元の文字コード・エンコーディングは Unicode の UTF-32 と UTF-16 だけであり、変換先は JIS2004 字形の横書きだけである。つまり Unicode ではなく JIS コードで DVI ファイルが出力される pTEX では使うことができない。upTEX は Unicode で DVI ファイルが出力されるので、これらを正しく設定することができれば使用できるが、JIS2004 字形の横書きしか使うことができない上、一部のクォートで文字幅の問題が発生し回避できない[26)27)]。さらに、一部の文字幅が AJ1 と異なっているため、そういった文字を使うと問題が発生することがある[28)]。また、AI0 であるがゆえに、フォントごとに CMap リソースのインストールと設定が必要な上、フォントのバージョンが変わるとマッピングが変更される可能性があり、新しい CMap リソースへの入れ替えを要するため、非常に煩雑である。

2.3.3 cmap テーブル

cmap テーブルを使う方法では、AJ1 フォントと同様に利用可能である。つまり、LuaTEX や XETEX でも、LilyPond でも AJ1 と同様に利用でき[29)]、字形の切り替えや縦書きも AJ1 と同程度に使うことが

23） Registry と Ordering の双方が一致している必要がある。Supplement は一致しなくても、使用する CID がフォントに含まれていれば問題ない。

24） 源ノフォントは言語ごとに字形を切り替えるため、CMap リソースが言語ごとに分かれている。

25） UTF-32 は master branch に、UTF-16 は release branch の Resources ディレクトリにある。

26） “UniJIS-UTF16-H” などと同様、upTEX で全角幅が前提のクォート 4 文字がプロポーショナル幅のグリフにマッピングされているが、全角幅にマッピングする CMap リソースが無くて回避できない。

27） upTEX で DVI ドライバに dvipdfmx を使う場合は、後述の cmap テーブルを使う設定にすれば、これらの問題は発生しない。

28） JFM の文字幅と源ノフォントの文字幅が異なると問題が発生する。

29） AJ1・AI0 ともに、フォント名もしくはフォントファイル名と、字形など OpenType feature の指定をする程度。

できる[30]。ただし、AJ1 ではないため luatexja-otf パッケージの機能は一部制限される[31]。

upTEX で DVI ドライバに dvipdfmx を使う場合は、設定で CMap リソースではなく cmap テーブルを使うようにする[32][33]ことができ、字形の切り替えや縦書きも含めて利用可能である[34][35]。ただし、AJ1 を前提とした OTF パッケージの機能は大部分が利用できない。また、AJ1 と文字幅が異なる文字を使用した場合には問題が発生する。

3 源ノフォントを使う

ここまでに、源ノフォントを含む AI0 のフォントが、一部の TEX エンジン・DVI ドライバや Ghostscript で使用困難であることを示した。使用困難となるのは、pTEX を使った場合と upTEX でも DVI ドライバに dvips など dvipdfmx 以外を使った場合である。これらについて、問題を回避して何とか源ノフォントを使う方法を考えてみる。

3.1 pTEX で使う

モダンな LuaTEX や XETEX に移行できず、upTEX にすることもできないが、源ノフォントを使いたい、というケースが考えられる。例えば、`ipsj.cls` [13] のように pLATEX のみの対応で、他の TEX エンジンに対応していないクラスファイルを使用する必要がある場合などである。こうした場合には、CMap リソースを新しく作成する方法、VF を使う方法、DVI ファイルを書き換える方法によって、源ノフォントを利用できるようにする[36]ことができる。

3.1.1 CMap リソースを作成する方法

pTEX は、出力する DVI ファイルが JIS コードだが、源ノフォントの CMap リソースに、変換元が JIS コードのものが無くて使用困難であった。ただ、そういった CMap リソースを作成することは可能であり、それを利用して pTEX で源ノフォントを使用する試み [14] がある。これは、源ノフォントの cmap テーブルをもとに CMap リソースを生成しているので、JIS2004 字形で横書きの CMap リソースだけしかなく、クォート文字幅の問題も発生する。そういった問題を無視できれば、dvipdfmx で PDF 生成できるし、dvips で生成した PostScript ファイルを Ghostscript で処理して PDF 生成することもできる。

JIS90 字形や縦書きの CMap リソースを作成することは可能[37]と考えられる。また、クォート文字幅の問題も CID を差し替えることで解消できると思われる。これによって、`.tex` ファイルをそのまま何も変更せず、pTEX で処理した DVI ファイルから源ノフォントを利用した PDF を生成することができる。しかし、AJ1 と文字幅が異なっている文字を使用すると、dvipdfmx で後続文字の位置がおかしくなるなど、組版に影響を及ぼしてしまう。また、フォントごとに CMap リソースを作成しなければならず、設定も非常に煩雑になってしまう。

3.1.2 VF を使う方法

pTEX が出力した DVI ファイルは JIS コードだが、これを VF (Virtual Font) の仕組みを使って Unicode へ変換することによって、upTEX 向けの設定を経由して源ノフォントを使う方法がある。これを

30） LuaTEX では luatexja-preset パッケージで源ノフォントを設定することも可能。

31） Unicode で表現できないものは使えない。\CID による直接指定は、漢字の異字体程度ならば IVS で代替される。

32） dvipdfmx のエンコーディング設定を CMap リソース名でなく “unicode” にすると、CMap リソースを経由しない Unicode 直接指定となり、cmap テーブルが使われる。

33） TEX Live 2018（一部 TEX Live 2017）以降が必要。

34） TEX Live 2019 で、kanji-config-updmap コマンドを用いて源ノフォントを使用した dvipdfmx の設定ができるようになった。

35） upTEX で全角幅が前提のクォート 4 文字は、これらだけを VF で割り当てた JFM “*-hq” に対して、OpenType feature で全角幅 “fwid” を設定し、全角幅の CID を得ることで、文字幅問題を回避している。

36） 他にも、DVI ドライバを拡張するという方法も考えられるが、本稿では触れない。

37） GSUB テーブルの “jp90” や “vert” をもとに CID を差し替えればよさそうである。

実装した PXufont パッケージ [5] は、まず、.tex ファイル中で PXufont パッケージを使うことで、pTEX から見える JFM を jis や jisg などから zu-jis や zu-jisg といった、“zu-” という名前で始まるものに置き換える。これによって、DVI ファイル中に現れる JFM 名が jis などから zu-jis などに変わることになる。次に、dvipdfmx が DVI ファイル中の JFM 名から zu-jis.vf などを読み込む。これらの VF は、JIS コードから Unicode への変換が行われるとともに、置き換え先の JFM に upTEX 用の uprml-h や uprml-hq が指定されている。その結果、upTEX が出力した DVI ファイルで VF を処理した後と同じ状態となる。後は、upTEX 用の dvipdfmx 設定に従って源ノフォントが使用される。

残念ながら .tex ファイルを何も変更せずに適用することはできないし、ipsj.cls は PXufont パッケージでフォント名を置き換えることができず失敗 [15] してしまう。AJ1 と文字幅が異なっている文字を使用した場合の問題も残る。

3.1.3 DVI ファイルを書き換える方法

.tex ファイルを一切書き換えず、pTEX が出力した DVI ファイルを書き換えることで源ノフォントを利用できる。DVIasm [16] は、DVI ファイルを逆アセンブルしてテキストファイルへ書き出すこと、テキストファイルをアセンブルして DVI ファイルを作成すること、ができるツールであり、pTEX の DVI ファイルにも対応している。DVI ファイルを逆アセンブルすることでデバッグに資することができる有用なツールであるが、テキストファイルからのアセンブルもできるため、DVI ファイルを書き換えるような用途にも有用である。

この方法で、pTEX が出力した DVI ファイル中の JFM 名 jis および jisg を、zu-jis および zu-jisg に書き換える[38)] シェルスクリプトの例を示す。

```
#!/bin/sh

INPUT_DVI_FILE=$1
OUTPUT_DVI_FILE=$2

TMP_FILE=`mktemp`

dviasm --ptex ${INPUT_DVI_FILE} | \
    sed -e 's/^fntdef: jis (/fntdef: zu-jis (/g' | \
    sed -e 's/^fntdef: jisg (/fntdef: zu-jisg (/g' | \
    sed -e 's/fnt: jis (/fnt: zu-jis (/g' | \
    sed -e 's/fnt: jisg (/fnt: zu-jisg (/g' \
        > ${TMP_FILE}

dviasm --ptex --output=${OUTPUT_DVI_FILE} ${TMP_FILE}

rm ${TMP_FILE}
```

これで出力した DVI ファイルは、JFM 名が zu-jis などに書き変わっているため、PXufont パッケージの VF によって Unicode 化され、upTEX の設定が使われるようになる。よって、pTEX でも .tex ファイルを書き換えず、また、ipsj.cls のような PXufont パッケージでフォント名を置き換えることができない場合でも、源ノフォントを利用することができるようになる。ただし、AJ1 と文字幅が異なっている文字を使用した場合の問題は残る。

3.2 upTEX で使う

upTEX では、DVI ドライバが dvipdfmx であれば（AJ1 文字幅の問題に触れる文字を使わない限り）源ノフォントを使用できるが、dvips の場合は困難である。源ノフォントの CMap リソースは配布されているので、これらを使えば dvips を利用できなくはないが、字形の切り替えや縦書きの対応ができず、クォート文字幅の問題が発生してしまう。とはいえ、pTEX と同様、これらを解消した CMap リソースを作成することは可能だと考えられる。しかし、AJ1 文字幅の問題は解消できないし、フォントごとに CMap リソースを作成しなければならず、設定が非常に煩雑という点も同様である。

38） DVIasm は、--ptex オプションがあると DVI ファイルの set2 が JIS コード、無ければ Unicode とみなされ、テキストファイルの UTF-8 と相互変換する。これを利用し、逆アセンブル時は --ptex オプションを付与し、アセンブル時には付与しないようにすると、JIS コードの DVI ファイルから Unicode の DVI ファイルへ変換することができる。ただし、JFM 名も Unicode 用に書き換えなければならないが、upTEX 標準の upjisr-h 系は pTEX の jis 系とメトリック非互換で使えない。古い ujis 系ならば概ね使えるが、クォート 4 文字が VF で pTEX の rml 系に割り当てられ、源ノフォントにすることができない。

3.3 TEX エンジンを移行する

pTEX にしか対応していないクラスファイルに手を入れるなどして、源ノフォントが使える TEX エンジンに対応させて移行する方法である。

3.3.1 新規作成する

レイアウトやフォーマットが厳密には決まっていないような場合には、クラスファイルを新規に作成してしまう方法が取り得る。FIT 情報科学技術フォーラム [17] では、過去に pLATEX 用テンプレートファイルが配布されていたようだが、現在は存在しない。また、原稿は PDF での提出であり、基本フォーマット（用紙サイズ、マージン、フォントなど）を厳守すれば、どのように作成してもよい、とされている。そこで、基本フォーマットをもとに、LuaLATEX に対応したクラスファイルを作成 [18] することで、源ノフォントを利用することができた[39]。

3.3.2 改変する

前述の通り、`ipsj.cls` は pLATEX のみ対応している。これを改変して upLATEX 用にする試み [19] をした。これは pLATEX で組版した場合と似たような組版結果になることを目指したが、不具合もある[40]。また、研究報告向ならば、原稿は PDF 提出なので、この改変クラスファイルを使う余地はあるかもしれないが、論文誌向けならば、PDF 提出は査読用だけであり、最終原稿では `.tex` ファイルなどを提出する[41]ことになるので、改変クラスファイルを使うことはできない。

4 原ノ味フォント

ここまで示してきたように、源ノフォントは一部の TEX エンジン・DVI ドライバや Ghostscript では使用困難だったり設定が煩雑であるほか、OTF パッケージのような AJ1 フォントが前提となるものは、ほぼ使うことができない。また、PDF からのテキスト抽出に問題が発生するケースもある。これらの大部分は源ノフォントが AI0 フォントであるがゆえに発生する問題であり、AJ1 フォントであれば発生しないものである。

そこで、AI0 である源ノフォントを AJ1 になるように組み替えた、「原ノ味フォント」[7] を制作した。「原ノ味」というのは、源ノフォントからグリフやテーブルが抜けていることを表すために「ゞ（さんずい）」を取り、AJ1 をもじって AJI にして、音から「味」という字をあてたものである。本フォントは源ノフォントの派生フォントになるため、SIL Open Font License 1.1 [3] が適用される。

4.1 制作の経緯

筆者は `ipsj.cls` [13] で源ノフォントを使いたいと考え、様々な方法を試行してきたが、これまでに述べてきたような問題が発生し、一筋縄ではいかなかった。AJ1 フォントであれば問題ないのだが、残念ながら筆者の環境にはこの用途に適した AJ1 フォントが無い[42]。一方、源ノフォントは SIL Open Font License 1.1 に基づくオープンソースのフォントである。つまり、一定の条件の下で改変や再配布が認められている。よって、源ノフォントのグリフを使った AJ1 フォントを作ること、そしてそれを再配布することは可能ではないか。そう考えて原ノ味フォントの制作を開始した。

39） 筆者は FIT2018 向けに、このクラスファイルと源ノフォントを使い、LuaLATEX で組版した原稿を投稿した。

40） 古い ujis 系を使っているため、クォート 4 文字が源ノフォントにならない。

41） プロが組版して論文誌に掲載される。

42） 唯一、Acrobat Reader に付属している小塚フォントはあるが、TEX を含む他ソフトウェアでの利用はライセンス的に不可と思われる。

4.2 搭載グリフ

源ノフォントが搭載している、かつ、AJ1 への対応が取れたものを搭載している。

- 漢字
  - Adobe-Japan1-6 漢字グリフすべて搭載
    - ルビ用の「注」
      U+6CE8 U+E0102 (AJ1 CID+12869)
      は非搭載[43)]
  - JIS X 0208 漢字グリフすべて搭載
  - JIS X 0213 漢字グリフすべて搭載
- 非漢字
  （ひらがな、カタカナ、英数字、記号類）
  - JIS X 0208 横書きグリフすべて搭載
    - JIS90 字形（CMap リソース “H”）
    - JIS2004 字形（CMap リソース “2004-H”）
  - JIS X 0208 縦書きグリフは以下の 4 グリフを除きすべて搭載（例示は横書きグリフ）
    - JIS90 字形（CMap リソース “V”）
      - 「‖」01-34, U+2016 (AJ1 CID+7895)
        ‘DOUBLE VERTICAL LINE’
    - JIS2004 字形（CMap リソース “2004-V”）
      - 「‖」01-34, U+2016 (AJ1 CID+7895)
        ‘DOUBLE VERTICAL LINE’
      - 「°」01-75, U+00B0 (AJ1 CID+8269)
        ‘DEGREE SIGN’
      - 「′」01-76, U+2032 (AJ1 CID+8273)
        ‘PRIME’
      - 「″」01-77, U+2033 (AJ1 CID+8283)
        ‘DOUBLE PRIME’
  - JIS X 0213 非漢字グリフには抜けあり
  - その他 AJ1-6 非漢字グリフには抜けあり

抜けているグリフの CID にはダミーグリフ（□の中に×が入ったような形）が入っている[44)]。上記で具体的に記載した非搭載グリフ（ルビ用 1 グリフ、非漢字縦書き 4 グリフ）は、いずれも源ノフォントが搭載していないため、原ノ味フォントに搭載できないものである。

非漢字の一部に、源ノフォントの AJ1 文字幅問題を解消するため、文字幅を強制的に AJ1 に合わせた[45)]ものがある。該当のグリフを使うと、左に寄って表示されたり、前後の文字に重なったり、不格好な表示になることがあるが、源ノフォントで問題となるような、他のグリフの位置に影響を及すことはなくなっているはずである。

4.3 生成プログラム

単純にフォントファイルだけを公開するのではなく、高品質なフォントをオープンソースで公開した Adobe や Google をはじめとする関係各位に敬意を表する意味でも、原ノ味フォント生成プログラム [8] をオープンソースライセンス[46)]で公開することとした。本稿ではプログラムの概要を説明する。詳細はドキュメントやソースを参照いただきたい。

4.3.1 動作

まず、源ノフォントの OTF ファイルを fonttools [20] の ttx で XML にし、C++で XML や CMap リソースなどから CID の対照表を作り、C++や sed で変換、最後に再び ttx で OTF ファイルを生成する、という方法を採った。C++は C++11 対応コンパイラ、XML ライブラリは pugixml [21]、その他 GNU Make に sed や sh などがあれば動作させることができる。

4.3.2 CID の対応

源ノフォントは AI0 フォントであり、CID の並び方が AJ1 と異なる。AJ1 化するためには AI0 CID → AJ1 CID 対照表が必要となる。源ノフォントは日本語用 (Region-specific Subset OTF) であっても、2 万近い数のグリフがあるため、人手で対照表を

43） AJ1 の「漢字グリフ」範囲外で漢字扱いではない。
44） 原ノ味フォント 20190501 以降の仕様。
45） 原ノ味フォント 20190824 以降の仕様。
46） 生成したフォントは SIL Open Font License 1.1 だが、生成プログラムは BSD 2-Clause である。

作るのは困難である。そこで、自動的に対照表を作るようにした。

■Unicode を介した変換　源ノフォントの cmap テーブルは Unicode → AI0 CID という変換表である。一方、CMap リソース “UniJIS2004-UTF32-H” は Unicode → AJ1 CID という変換表である。そこで、cmap テーブルを逆変換して、AI0 CID → Unicode とし、さらに CMap リソースの変換を重ねて、AI0 CID → Unicode → AJ1 CID という変換にすることで、AI0 CID → AJ1 CID の対照表を作っている。

この方法は、源ノフォント以外の AI0 フォントでも利用することができる。また、源ノフォントを AJ1 以外（つまり日本語以外）の ROS に組み替える際にも、同様に利用することができるだろう。

■漢字　Unicode を介した変換だけでは、漢字の異字体などに対応できない。だが、源ノフォントでは `aj16-kanji.txt` という AJ1 CID と源ノフォントのグリフ名の対応表が配布されている。さらに、`AI0-SourceHanSerif` と `AI0-SourceHanSans` という、AI0 CID とグリフ名の対応表も配布されている。これらをつなげば AI0 CID → AJ1 CID の対照表を作ることができる。ただし、漢字グリフしか含まれていない。

この方法は、源ノフォントのみ、しかも AJ1 に対してだけ利用することができるが、異字体や JIS90 字形・JIS2004 字形も含め、全漢字グリフの対照表が得られるので、非常に強力である。

■縦書きなど　源ノフォントの GSUB テーブルから、縦書き用 OpenType feature “vert” を読み込むと、横書き用 AI0 CID →縦書き用 AI0 CID という変換表が得られる。AJ1 の GSUB も配布されている [4] ので、ここから、横書き用 AJ1 CID →縦書き用 AJ1 CID という変換表が得られる。前者を逆変換して、AI0 CID → AJ1 CID 対照表を介してつなげると、縦書き用 AI0 CID →横書き用 AI0 CID →横書き用 AJ1 CID →縦書き用 AJ1 CID となり、縦書き用の対照表が得られる。

縦書き “vert” 以外にも、CID が 1 対 1 対応になっている、全角幅 “fwid”、半角幅 “hwid”、プロポーショナル幅 “pwid”、ルビ “ruby” についても、同様の方法で対照表を作ることができる。

4.3.3　OpenType テーブルの変換

ほとんどのテーブルを変換している。フォント名称などの置き換え、対照表に従った AI0 CID → AJ1 CID への置き換え、置き換えることができなかった AI0 CID の削除、欠番になっている AJ1 CID へのダミーグリフ挿入、AJ1 文字幅への変更などを実施している。

4.4　pTEX・upTEX で使う

源ノフォントと異なり比較的簡単に使うことができる。フォントファイルとマップファイルを適切に配置すれば普通に使える。ただし搭載しない（抜けている）グリフを使うことはできないし、文字幅を変更したグリフで表示が不格好になるものもあるので、4.2 節「搭載グリフ」や原ノ味フォントのドキュメントなどを参照して、ご注意いただきたい。

4.4.1　マップファイル

マップファイルは原ノ味フォントの配布サイト [7] にて配布している。適切に配置して TEX Live の kanji-config-updmap コマンドを使えば、dvipdfmx と dvips 双方で原ノ味フォントを使うことができるようになる。

4.4.2　`ipsj.cls`

`ipsj.cls` には、著者が組版するときのためのクラスオプション “submit” がある。“submit” が無ければ、プロの環境用として和文フォントに明朝・ゴシックそれぞれ 2 ウェイトずつ使う。“submit” を付けると、pTEX 環境であれば必ず使えるであろう、それぞれ 1 ウェイトずつしか使わないよう

になる。[47][48]。以下の dvipdfmx 用マップファイルは、プロ環境用の和文 2 ウェイトずつ、全 4 フォントを原ノ味フォントで代替する設定である。クラスオプション “submit” を外してこのマップファイルを使うと、見た目が論文誌に掲載されている、プロが組版した論文に近くなる。

```
rml             H       HaranoAjiMincho-Light.otf
gbm             H       HaranoAjiGothic-Regular.otf
futomin-b       H       HaranoAjiMincho-Regular.otf
futogo-b        H       HaranoAjiGothic-Medium.otf
```

なお、論文誌向け（クラスオプション “techrep” 無し）で使う場合は、著者は最終原稿として `.tex` ファイルを提出し、プロが本物のフォントを使って組版するため、本設定は関係ない。最終原稿前の査読用については、著者が組版し PDF 提出するため、本設定が使えるかもしれないが、フォントの違いが査読に影響することはないはずであり、あくまでも自己満足の域に過ぎないと思われる。一方、研究報告向け（クラスオプション “techrep” 有り）で使う場合は、最終原稿が著者組版の PDF 提出なので、本設定が使えるかもしれないが、そもそも見た目が論文誌のように統一されておらず、どれが本物に近いとは言い難く、これも自己満足に近いと思われる。

4.5 その他の環境で使う

LuaTEX や XeTEX でも、LilyPond でも、他のフォントと同様に利用可能である。IVS や OpenType feature も使うことができる。Ghostscript でも他の AJ1 フォントと同様の設定で利用可能である。

ただし、pTEX や upTEX で使う場合と同様、使えないグリフや表示が不格好なグリフがあるので、4.2 節「搭載グリフ」やドキュメントなどを参照して、ご注意いただきたい。

47） 欧文フォントも Helvetica を使わなくなるなど少し変わるようだが、今どきの環境であれば、わざわざ除外しなくても Nimbus Sans で代替できるだろう。

48） クラスオプション “submit” 有無は、論文誌向け・研究報告向け、どちらでも同様の動作である。つまり、クラスオプション “techrep” 有無とは独立の設定である。

5 ToUnicode CMap

ToUnicode CMap は PDF からテキスト抽出する際に使われる、いくつかある機構のうちの一つである。PDF で CID-keyed フォントが使われていて、エンコーディングが Identity-H や Identity-V である場合[49]、PDF 中の文字は CID で指定されており、Unicode のような文字コードは失われている。テキスト抽出するには、使われている CID がどの Unicode に対応するか、というテーブルが必要となる。これが ToUnicode CMap である。

5.1 テキスト抽出

AJ1 フォントの場合は、PDF 内に ToUnicode CMap が埋め込まれていなくても、PDF viewer が適切な AJ1 の ToUnicode CMap を持っていれば[50]、テキスト抽出可能である。一方、AI0 フォントの場合は、CID と文字の関係がフォントごとに異なるため、PDF に ToUnicode CMap が埋め込まれていないとテキスト抽出できない。

5.2 ToUnicode CMap の自動生成

pTEX や upTEX で使われる dvipdfmx や、XeTEX で内部的に使われる xdvipdfmx は、AJ1 フォントに対しては ToUnicode CMap の生成や埋め込みをせず、AI0 フォントに対しては生成して埋め込む、という動作になっており、いずれのフォントでもテキスト抽出できるよう配慮されている。また、LuaTEX は、フォントに関係なく ToUnicode CMap を生成して埋め込む[51]ため、テキスト抽出は可能である。一方、Ghostscript は、PDF を出力する際はフォントに関係なく ToUnicode CMap の生成や

49） dvipdfmx で CID-keyed フォントを使って生成した PDF はそうなる。

50） CJK をまともに扱うことができる PDF viewer であれば持っているはずである。持っていないものでも追加インストールなどで対応可能なものが多いと思われる。

51） さらに、フォントが AJ1 であっても AI0 に書き換えてしまう。

埋め込みをしない。そのため、AJ1 フォントはテキスト抽出できるが、AI0 フォントはテキスト抽出できないということになる。なお、LilyPond は PDF を出力する際、内部的に Ghostscript を使用している。そのため、Ghostscript 同様、AJ1 フォントはテキスト抽出できるが、AI0 フォントはテキスト抽出できない。

5.2.1 逆変換による自動生成

TEX Live 2019 の dvipdfmx（XETEX で使われる xdvipdfmx も含む）や LuaTEX は、使用した CID で cmap テーブルを参照し、この逆変換で ToUnicode CMap を生成する。この方法は DVI の文字コードが何であっても動作するが、逆変換に起因する問題が発生してしまう。

cmap テーブルは、表 1 にある「見」のように、複数の Unicode に対して同一の CID を割り当てている場合があり、逆変換の際にはどちらの Unicode を使うか判断する必要がある。どちらがよりふさわしいかは個々の CID によって異なるため個別に判断する必要がある。例えば、「見」の場合は、通常の漢字として使われるのは U+898B なので、これを選択するべきであるが、間違っていると意図しないテキスト抽出結果になってしまう。

また、表 1 では、「𠮟」のような縦書き用のグリフや JIS2004 字形・JIS90 字形でデフォルトでない方のグリフ（JIS2004 字形のがデフォルトの源ノフォントならば、「飴」のような JIS90 字形のグリフ）は、そもそも cmap テーブルに存在しないので逆変換が生成できず[52]、テキスト抽出の際に抜けてしまう。

さらに、縦書き用のグリフなどのデフォルトではないグリフが、デフォルトのグリフとは別の Unicode からマップされている場合がある。例えば、開き括弧「(」U+FF08 は縦書きでは「﹇」になるが、これは U+FF08 とは別に U+FE35 からマップされている。そのため、縦書き中の開き括弧を含んだ文字列をテキスト抽出した場合、本来であれば U+FF08 が抽出されるべきだが、U+FF35 になってしまい、結果がおかしくなる。

5.2.2 入力からの自動生成

TEX Live 2018 の dvipdfmx は、DVI の文字コードが Unicode であると仮定し、これを元に ToUnicode CMap を生成していた。この方法では逆変換を行わないため、逆変換が原因の問題は発生しない。pTEX の場合は JIS コードなのでこの仮定は当てはまらないのだが、AJ1 であれば ToUnicode CMap が生成されないので問題ない[53]。しかし、源ノフォントを JIS コードの CMap リソースを作って使った場合や、VF で JIS コードから Unicode へ変換した場合には、AI0 フォントなので ToUnicode CMap が自動生成されるが、入力が Unicode ではないため不正な ToUnicode CMap となり、テキスト抽出の結果が派手に文字化けしてしまう。この挙動は dvipdfmx 20180902 で逆変換による自動生成に変更されており、TEX Live 2019 では文字化けしないようになった。

5.3 調整済の ToUnicode CMap

ToUnicode CMap の自動生成には、いくつか問題があることを示した。そこで、こういった問題が発生しないよう、調整済の AJ1 用 ToUnicode CMap である、“Adobe-Japan1-UCS2” がある [11]。複数の Unicode に対して同一の CID を割り当てている場合でも、よりふさわしい Unicode が選ばれ、縦書きやデフォルトではない字形でも抜けることはなく、別の Unicode からのマッピングがあるような CID でも、よりふさわしい Unicode の方が得られるように調整されている。

PDF に AJ1 フォントの ToUnicode CMap が埋め込まれていない（その分、ファイルサイズを小さ

52） GSUB テーブルには存在するはずなので、cmap テーブルで見つからなければ、GSUB テーブルで探してデフォルトの CID を見つけ、改めて cmap テーブルで探す、という方法を取れば見つけられるはずではある。

53） TrueType の場合は GID の並びが AJ1 相当に並び替えられ、ToUnicode CMap が生成されないようである。

くすることができる）状態でも、PDF viewer がこの “Adobe-Japan1-UCS2” を持っていれば[54]テキスト抽出ができるし、自動生成に伴う問題も発生しない、というわけである。

5.4 PDF/A-2u

PDF/A 規格 (ISO 19005) は、遠い将来の PDF viewer でも、現在と同様の見た目で表示したり、利用したりできる PDF を実現することを目指していて、通常の PDF 規格をベースに、追加で守らなければならない事項が定められている。PDF/A-2u は、そのうちの一つであり、テキスト抽出ができる（Unicode が取得できる）ことが必須事項の一つになっている。

5.4.1 PDF/A 作成

例えば、LuaTEX で pdfx パッケージ [22] を使うことで PDF/A に準拠した PDF を生成することができる。本稿でもこれを利用して PDF/A-2u 規格に準拠した PDF としている。しかし、pdfx パッケージは、出力する PDF に対して、「PDF/A である」というメタデータを埋め込むことなどはできるが、単純にそれだけでは PDF/A に準拠したことにはならない。例えば、PDF/A では許されない条件の PDF [55]を`\includegraphics` で取り込んだ場合など、これを PDF/A に準拠するよう変換することはできない。

5.4.2 PDF/A バリデーション

「PDF/A である」というメタデータを持つ PDF を PDF viewer で開いた場合、「PDF/A 規格に準拠している可能性があり」というように、「準拠している」とは言い切らない表示をするものが多い。PDF/A を名乗っているだけで準拠していない場合は、現在の PDF viewer では一見正常に取り扱うことができるように見えても、将来の PDF viewer ではおかしくなってしまうかもしれない。そこで、本当に PDF/A に準拠しているか、バリデーションを行う必要がある。オープンソースの PDF/A バリデーションツールとして、veraPDF [23] がある。本稿は、この veraPDF で PDF/A-2u に準拠することを確認しながら作成している。

5.4.3 テキスト抽出の条件

本来であれば ISO の規格書をあたるべきであるが、有償であるため veraPDF のドキュメントをあたることにする。この Rule 6.2.11.7-1 [24] を読むと、基本的には ToUnicode CMap が必要だが、いくつか例外的に無くてもよい条件があり、そのうちの一つが AJ1 となっている。AJ1 であれば、PDF に ToUnicode CMap が埋め込まれてなくても、PDF viewer が “Adobe-Japan1-UCS2” を持っていればテキスト抽出できるため、このような規定になっていると思われる。いずれにせよ、規格上も AJ1 であれば ToUnicode CMap が無くても、テキスト抽出できるとみなしていることがわかる。

6 LuaTEX と ToUnicode CMap

dvipdfmx や xdvipdfmx は AJ1 フォントであれば ToUnicode CMap を生成せず、埋め込むこともない。しかし、LuaTEX は AJ1・AI0 に関わらず ToUnicode CMap を生成して埋め込み、しかも埋め込むフォントの ROS が AJ1 [56]でも、AI0 へ書き換えてしまう[57]。この ToUnicode CMap は逆変換で自動生成されたものなので、これまでに述べてきたような問題を抱えている。

54） PDF viewer によっては、“Adobe-Japan1-UCS2” そのものではなく、これをなにがしかの形式に変換したテーブルを持っている場合もある。

55） RGB と CMYK の双方を一つの PDF 内で使うことができない、など。

56） フォントファイル中の/CIDSystemInfo 辞書に ROS が AJ1 である旨が記載されている。

57） PDF に埋め込まれたフォントは、ROS が AI0 である旨を/CIDSystemInfo 辞書に記載される。

6.1 ToUnicode CMap 削除

AJ1 から AI0 へ書き換えたからと言って、CID が変わるわけではない[58]。そのため、PDF に埋め込まれた原ノ味フォントの ROS を AI0 から AJ1 に戻し、ToUnicode CMap を削除することで、逆変換による問題を解消できるはずである。そこで、qpdf [25] を使い、手動でこのような加工を行ったところ、想定通りとなった[59]。

6.2 削除ツール

この ToUnicode CMap 削除を自動的に行うツールとして pdf-rm-tuc [9] を作成した。PDF ライブラリとして libqpdf を使用しており、一部 qpdf と同様のコマンドラインオプションを使うことができる。これにより、`--linearize`（Web 表示用に最適化）、`--object-streams`（オブジェクトストリーム設定）、`--newline-before-endstream`（PDF/A 向け）、`--qdf`（QDF 出力）が使用できる。本稿の PDF も、このツールを使って PDF/A-2u を維持したまま ToUnicode CMap を削除したものである。本稿は、LuaTEX の出力した PDF ではファイルサイズが 362 KB 程度であったが、本ツールを使って ToUnicode CMap の削除などを行うことにより、303 KB 程度に低減することができた。

なお、本稿も含め、原ノ味角ゴシックで「𠮟」「𠀋」を使っていた場合、削除ツールで ToUnicode CMap を削除すると、現行の veraPDF 1.14 ではバリデーションに通過しなくなる。これは、veraPDF に内蔵されている “Adobe-Japan1-UCS2” が AJ1-6 のものであり、AJ1-7 で追加されたこれらの文字を使っていると Unicode を得ることができないため、と思われる。veraPDF のインストールディレクトリ内の `bin/greenfield-apps-(version).jar` 内部にある “Adobe-Japan1-UCS2” を AJ1-7 のものに置き換えることで、バリデーションに通過するようになる。また、veraPDF 内の “Adobe-Japan1-UCS2” を更新する pull request を投げたところ、受け入れられた [26] ので、将来の veraPDF では、わざわざ置き換える必要はなくなると思われる[60]。

7 おわりに

源ノフォントは AI0 フォントであるため、日本語フォントが AJ1 であることを前提としたシステムで利用することは困難である。本稿では、まず、AJ1 と AI0 を比較し、一部のシステムで源ノフォントが利用困難となる理由を述べた。そして、これを解決するため AJ1 に組み替えた原ノ味フォントについて述べ、源ノフォントでは困難であった利用方法を紹介した。次に PDF からのテキスト抽出に使われる ToUnicode CMap について説明し、源ノフォントでは問題になる場合があることを示した。最後に、テキスト抽出にも関わる PDF/A-2u 規格について述べ、原ノ味フォントを使って LuaTEX が出力した PDF/A-2u 準拠の PDF から、準拠のまま ToUnicode CMap を削除できるツールを紹介した。

原ノ味フォントのリリース当初は、AJ1 だが抜けているグリフがあるので、それを埋めることが課題であると考えていた。そのためには、源ノフォントの AI0 との紐づけを改良していけばよいと考えていた。最初は、この改良でグリフ数を増やし、抜けを埋めていくことができた。しかし、ある程度進んだところで、どうやらそもそも源ノフォントには存在しないグリフが必要になりそうだ、ということがわかってきた。グリフの「変形」ができれば、90 度回転で縦書き用、縮小してルビ用、疑似的にイタリック用、などのグリフを作る

58） CID = GID の場合。CID ≠ GID の場合は、フォント埋め込み時に元の GID を埋め込む際の CID として使うようにしているらしく、結果として CID が変わってしまう。

59） 当初、原ノ味フォントは CID ≠ GID であったため、この加工ではテキスト抽出が文字化けしてしまった。原ノ味フォント 20190501 で CID = GID となるよう仕様変更したため、文字化けせずにテキスト抽出可能となった。

60） PDF/A-2 規格制定時に存在しなかった AJ1-7 を使ってよいのか、という疑問は残る。

ことができるかもしれない。もしかしたらルビ用などは、元のグリフをそのままルビ用の CID へコピーしてもいいかもしれない。プロポーショナル幅について決まりが無いのであれば、全角幅や半角幅のグリフをそのままプロポーショナル幅の CID へコピーしてもいいかもしれない。あまり手間を掛けたくはないが、悩みどころである。

原ノ味フォント 20190824 で急遽対策を施した AJ1 文字幅問題は、本稿執筆中に気が付いたものである。単に文字幅を強制的に上書きしただけなので、元々の幅が狭かったグリフは左に寄ってしまう。これを何とかするには CFF Charstring を編集する必要があり、かなり手がかかってしまう。中央配置になるようスライドさせるだけなら（ヒント情報を無視すれば）何とかなりそうな気もしているが、幅に合わせて拡大させるならば変形の必要がある。さらに、元々の幅が広かったグリフの場合は、縮小する変形をさせなければ幅の中に収まらない。変形させるとなれば Charstring ラスタライザ相当の処理が必要で、かなりの実装量となりそうである。ゴシックなら源ノ等幅などから持ってくる方法もなくはないが、他のフォントから持ってくるには Charstring のサブルーチン呼び出しを展開できる必要がありそうだし、そもそも全角幅ではないようなので、さらに変形が必要になるかもしれない。この問題は、そこそこ使用頻度が高そうに思えたギリシャ文字でも踏んでしまうのだが、実際には和文フォントのものはあまり使わない（数式フォントの方を使うことが多い）し、キリル文字にしろ、他の記号類にしろ、和文フォントで使うことはあまり無く、影響は限定的ではないだろうか、とも思っているので、どこまでやるか考えどころである。

また、本稿は、原ノ味フォントとは一見関係なさそうな PDF/A についても触れている。あちこちのシステムで AJ1 か AI0 かによって、テキスト抽出に関係する挙動に違いがあり、テキスト抽出に深く関係する PDF/A-2u を引っ張り出してきたものである。PDF の様々な規格について、例えば、印刷用の PDF/X については、あちこちで解説などを見ることができるが、長期保存用の PDF/A については、数が少ないように感じている。本稿が PDF/A を作成したい方の助けにもなるようであれば幸いである。他にも、PDF に対する電子署名を試してみようと考えたが、PDF/A を維持したまま電子署名できるフリーの PDF 処理系を見つけることができず、また、認証局が発行した電子署名用のクライアント証明書が必要だが有償のものが多く[61]、断念した。

原ノ味フォントの構想を最初に思い付いたときは、既に誰かが同様のフォントを作っているのではないかと思ったのだが、探してみるとどこにも無さそうであった。そこで、様々な調査をし、実装をし、動かしてみて、改良を進めてきた。始める前から薄々感づいてはいたのだが、実際に進めていくと、やはりフォント回りは歴史的経緯があるからなのか、かなり複雑怪奇になっているように思える。本稿は、それなりの調査をした上で執筆したつもりだが、簡潔にまとめることができず、アブストラクトとは言い難いような分量になってしまったため、不十分なところや間違いなどがあるかもしれない。本稿のソースファイルや関連ファイルはサイト [27] で公開し、正誤表や修正版なども公開することを検討したいので、お気づきの点があればご連絡いただけれるとありがたい。

（公開用アブストラクト 2019 年 8 月 30 日提出）

参考文献

[1] 源ノ明朝/Source Han Serif. https://github.com/adobe-fonts/source-han-serif.

[2] 源ノ角ゴシック/Source Han Sans. https://github.com/adobe-fonts/source-han-sans.

61) CAcert は無償で発行しているが、筆者は保証ポイントを持っておらず、名前入り証明書が得られない。

[3] SIL Open Font License 1.1. http://scripts.sil.org/OFL.
[4] Adobe-Japan1 文字コレクション. https://github.com/adobe-type-tools/Adobe-Japan1.
[5] PXufont Package — Emulate non-Unicode Japanese fonts using Unicode fonts —. https://www.ctan.org/pkg/pxufont.
[6] LilyPond — みんなの楽譜作成 —. http://lilypond.org/
[7] 原ノ味フォント. https://github.com/trueroad/HaranoAjiFonts.
[8] 原ノ味フォント生成プログラム. https://github.com/trueroad/HaranoAjiFonts-generator.
[9] Remove ToUnicode CMap from PDF. https://github.com/trueroad/pdf-rm-tuc.
[10] CMap Resources. https://github.com/adobe-type-tools/cmap-resources.
[11] Mapping Resources for PDF. https://github.com/adobe-type-tools/mapping-resources-pdf.
[12] font maps for dvipdfmx. https://github.com/texjporg/jfontmaps.
[13] 情報処理学会 LaTeX スタイルファイル. https://www.ipsj.or.jp/journal/submit/style.html.
[14] cmap から CMap を作る話. マクロツイーター, 2015-10-10. https://zrbabbler.hatenablog.com/entry/20151010/1444491218.
[15] https://twitter.com/zr_tex8r/status/1111935971627401217.
[16] DVIasm — A utility for editing DVI files —. https://ctan.org/pkg/dviasm.
[17] FIT2019 第 18 回情報科学技術フォーラム. https://www.ipsj.or.jp/event/fit/fit2019/.
[18] FIT2019 向け LaTeX クラスファイル. https://github.com/trueroad/FITpaper-class.
[19] ipsj.cls を upLaTeX 用にしてみる（源ノ明朝・源ノ角ゴシックを使う）. https://gist.github.com/trueroad/c44312923bf02226c2274388941d0453
[20] A library to manipulate font files from Python. https://github.com/fonttools/fonttools.
[21] pugixml — Light-weight, simple and fast XML parser for C++ with XPath support —. https://pugixml.org/
[22] pdfx — PDF/X and PDF/A support for pdfTeX, LuaTeX and XeTeX —. https://ctan.org/pkg/pdfx.
[23] veraPDF | Industry Supported PDF/A Validation. https://verapdf.org/.
[24] Rule 6.2.11.7-1, PDF/A-2 and PDF/A-3 validation rules. https://docs.verapdf.org/validation/pdfa-parts-2-and-3/#6.2.11.7-1.
[25] QPDF. https://github.com/qpdf/qpdf.
[26] Update ToUnicode CMaps from upstream. https://github.com/veraPDF/veraPDF-parser/pull/379.
[27] TeXConf 2019 一般講演「原ノ味フォントと ToUnicode CMap」関連資料. https://github.com/trueroad/tr-TeXConf2019.