Technical Report

UCAM-CL-TR-591
ISSN 1476-2986

Number 591

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Conversion of notations

Silas S. Brown

June 2004

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted November 2003 by the author for the degree of Doctor of Philosophy to the University of Cambridge, St John's College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/TechReports/

ISSN 1476-2986

Abstract

Music, engineering, mathematics, and many other disciplines have established notations for writing their documents. The effectiveness of each of these notations can be hampered by the circumstances in which it is being used, or by a user's disability or cultural background. Adjusting the notation can help, but the requirements of different cases often conflict, meaning that each new document will have to be transformed between many versions. Tools that support the programming of such transformations can also assist by allowing the creation of new notations on demand, which is an under-explored option in the relief of educational difficulties.

This thesis reviews some programming tools that can be used to manipulate the tree-like structure of a notation in order to transform it into another. It then describes a system "4DML" that allows the programmer to create a "model" of the desired result, from which the transformation is derived. This is achieved by representing the structure in a geometric space with many dimensions, where the model acts as an alternative frame of reference.

Example applications of 4DML include the transcription of songs and musical scores into various notations, the production of specially-customised notations to assist a sight-impaired person in learning Chinese, an unusual way of re-organising personal notes, a "website scraping" system for extracting data from on-line services that provide only one presentation, and an aid to making mathematics and diagrams accessible to people with severe print disabilities. The benefits and drawbacks of the 4DML approach are evaluated, and possible directions for future work are explored.

Acknowledgements

I would like to thank my supervisor Peter Robinson and the members of the Rainbow research group and the Computer Laboratory in general for their support during this work.

This work was sponsored by a studentship from the Engineering and Physical Sciences Research Council.

Contents

List of Figures

1 Introduction

1.1 Notations

The information age. The first electronic computers were valued for their ability to perform numerical computations quickly. But since Leavitt and Whisler coined the term "information technology" in 1958 [46], computers have increasingly been used for the storage and processing of information. Records and other documents have been kept in databases since the 1960s. Publishers have used information technology to assist with their typesetting, and as a side-effect they have accumulated "electronic" copies of their publications. The increasing presence of home and office computers in the 1980s heralded a correspondingly greater production of electronic information, and the invention of the World Wide Web by Tim Berners-Lee in 1992 [11] has encouraged the widespread use of the Internet for electronic publishing. In addition, there are continuing developments in the field of optical character recognition (OCR), enabling computers to scan and process printed documents that have not been stored electronically.

Notations. The information that is stored and processed by computers has not been limited to text. Besides the storage of signal-based data such as audio and video, computers have been used to process a significant amount of non-textual symbol-based data, such as musical scores, mathematics, schematics, and scientific models. Much attention has also been given to the presentation of human-readable computer program code, and to representing the writing systems of many different countries.

1.1.1 Symbolic vs signal-based representation

Representations of information in computer memory can generally be divided into two categories: symbolic and signal-based. Symbolic representations are based on the practice of encoding the *identifications* of symbols, whereas signal-based representations are based on *measurements* of some physical quantity (such as light or sound) which may contain information. Scanned images and recorded sounds are signal-based, whereas program code is largely symbol-based, or *symbolic*.

Hybrid. Some representations are symbolic in one sense and signal-based in another. For example, many desktop publishing applications *identify* which symbols are on the page but *measure* their positions and sizes, which the user can indicate with a pointing device—this is used to record presentational refinements that are more difficult to encode symbolically. However, algorithms that act on the *logical* relationships between items on the page, for example to transform the document into a completely different format, may need to *interpret* the physical positioning data, which can be difficult (although it can sometimes be ignored).

Another example is a MIDI file generated by a music keyboard. This *identifies* which notes are being played and also *measures* the velocity and timing—the measurements can record subtle artistic expressions, but if the music is to be converted to staff notation then the data must be interpreted.

Need for symbolic representation. Signal-based representations may lead to more accurate reproduction of an original source, but they limit the effectiveness of automatic processing—apart from rudimentary transformations on the signal itself (such as amplification), any processing must first involve *interpreting* the signal into a symbolic representation. Such signal-to-symbol conversion lies in the domain of "artificial intelligence" and is currently unreliable. It is beyond the scope of this thesis, which deals with symbolic representations.

Awareness. Users of applications that prepare documents are often unaware of whether or not some aspects of their information are being stored as measurements rather than codes, and the significance of this. User interfaces with direct manipulation and WYSIWYG (what you see is what you get) can make signals and symbols indistinguishable, so it may seem arbitrary that some documents can be processed while others cannot.

However, many experienced users wish to ensure that their documents can be transformed or otherwise processed automatically in future. Others wish to avoid recording unwanted inaccuracies. This is especially true for people with disabilities that hamper them from accurately positioning or timing their input, but it is also true for non-disabled people who need maximum accuracy, or who wish to utilise typesetting software that produces good presentation algorithmically. Consequently, much accurate work is represented symbolically; it is not unreasonable to limit this thesis to symbolic representations.

1.1.2 Structure

Purposeful, not aesthetic. If a building, a machine, an organism, a program, or anything else with an overall function, is *structured*, then it can be thought of as an assembly of different parts, each of which has a different purpose that contributes toward the overall functionality. Structure in works of art is usually aesthetic; it does not always make sense to assign different purposes to different parts of the work of art. When speaking of *structured data*, the author is referring to structure in the non-aesthetic, functional sense.

Structure in interpretation. The purpose of data is to describe something (a document, a work of art, a sequence of events, or something else) in the medium of computer memory, and each part of the data has the purpose of describing a particular part of whatever is being described. However, a computer has no consciousness; it cannot comprehend the semantics (meaning) of what its data is describing. Computers run programs and perform actions. Hence, to a computer, data is effectively 'structured' if the program treats it as such. In other words, this author defines data to be 'structured' if and only if *the program treats different parts of it in different ways.*

This clearly introduces a dependence on the program that is being used. So structured data is not always structured. A hexadecimal editor, for example, pays little attention to any structure of the data beyond such things as the number of bits in a byte (and possibly the number of bytes in a word). If that same data were loaded into another program, it might prove to have a much more complex structure.

1.2 Transformation

It is often necessary to transform data from one structure to another:

- Compilers and other software development tools are based on transforming the programmer's input into an executable form (or, in the case of generative programming, transforming it into lower-level code that is to be compiled).
- Different programs (and in some cases items of hardware) use different data formats to represent the same thing, so conversion is often needed when exchanging data between them. For example, there are many converters between different document formats, different sound and image file formats, musical score formats, and so on. Object request brokers (ORBs) and middleware frequently employ conversion.
- Some algorithms can be simplified if the data is first transformed into a convenient structure. Many algorithms can be regarded as transformations in their own right.
- Transformation can be important when presenting data to the user and when accepting user input.

This thesis is primarily concerned with the last point—the importance of transformation in user interaction, particularly when users have special needs. Previous transformation research has not addressed this application as much as it has addressed the others.

1.3 Diversity of special needs

A popular misconception is that all special needs are identical (or nearly so), and that they are adequately addressed by existing systems and hence there is no need of further research. This section aims to show otherwise.

Special needs are diverse and can be difficult to anticipate. A person's ability to use data in printed form may be hampered by a print disability, such as blindness, low vision, or dyslexia. It may also be hampered by educational and cultural differences—the person may have learned a notation that is different from the one being used in the presentation. Motor disabilities may limit data entry and interactive navigation with notations.

Diversity of print disabilities. The diversity of print disabilities is rarely acknowledged in the literature. Many papers use the phrase "blind and visually impaired" when discussing designs for blind people, implying that people with low vision work in the same way; in fact, many with low vision use their residual sight as much as they can [48]. Other papers (such as Hermsdorf et al [30]) assume that the needs of all partially-sighted people are much the same. Jacko et al [33, 34] show that this is not true, and suggest that a user's needs can be determined by clinical assessment. Even this can be difficult in the case of some eye conditions, such as nystagmus, which can vary over time and can produce different perceptual results for different people [67], or cortical visual impairment, which involves the brain's visual processing as well as the optical system. Some users need to be given control over the presentation themselves, as Gregor and Newell explain for some cases of dyslexia [28].

Many industry-standard applications already allow the customisation of fonts, sizes and colours, but it can be difficult and is not always reliable. Not all of these applications allow the *layout* of structured data to be changed to compensate for the reduced viewing area to text size ratio (a problem that is also prevalent in the mobile telephone industry). Alternative layout algorithms can also benefit people who have difficulty fixing their gaze, and this requires more customisation.

Specialist notations. Educational background—the type of things taught in the curriculum a person has taken—can contribute to the requirement for an alternative presentation, particularly in an international setting. A good example of this is in music—besides Western staff notation, musicians use various tablature and instrument-specific notations, as well as *sol-fa*, Chinese *Jianpu* notation, and others, and it is often possible to transcribe a piece of music from one notation into another in order to make it accessible to a greater number of musicians. Braille music also has numerous different versions across the world.

Notations can be customised for different tasks, such as sequential reading, rapid overview, or detailed analysis. Often it is desirable to omit or include certain details depending on how the document will be used, because people with print disabilities are frequently unable to skip over unwanted information at speed. Additionally, educational establishments can customise notations for pedagogical purposes.

Data entry is another task that can call for alternative notations, since optimising for input and/or editing is different from optimising for reading. Even direct-manipulation interfaces sometimes use a hidden input notation in their controls. For the sake of usability, there is usually some compromise so that the input and output notations are conceptually similar, but this can be overshadowed by a disabled user's accessibility needs. For example, someone with extreme typing difficulties might prefer a terser input notation even if it means more training.

Multiple and unforeseen needs. Although it is common practice to focus on one need at a time, it is possible to envisage individuals who have a combination of several of the above-mentioned special needs, as well as additional ones that have not been anticipated by research.

Cognitive disabilities. Many types of disability affect the brain's cognition. The term "cognitive disability" is frequently defined more specifically as a disability associated with cognition *in general* rather than the use of any particular notation, thereby excluding most types of dyslexia, which is regarded as a print disability (see above). Although this distinction is controversial, it is useful as an indicator of which disabilities the conversion of notations can reasonably be expected to address; its benefit to *general* cognitive disabilities would be very limited.

1.4 Transformation in universal design

Universal design aims to develop "technologies which are accessible and usable by all citizens. . . thus avoiding the need for *a posteriori* adaptations or specialised design" [62]. As Vanderheiden points out [72, 71], this does not mean designing one homogeneous user interface to fit everybody, since people have conflicting requirements and a "lowest-common denominator" interface would be limited indeed:

> "It can't have a *visual* interface [because some people can't see]; it can't have an *audio* interface [because some can't hear];. . . you could design a *brick*. . . "

Hence, a technology that fulfills (or partly fulfills) the aims of universal design is likely to include transformation functionality. This is not specialised design if it is customisable and extensible, and it is not *a posteriori* adaptation if the transformation needs were foreseen *a priori* in the original design.

1.5 Aims & objectives

The work described in this dissertation has developed a generalised transformation framework that can be applied to many different transformations, particularly within the area of converting notations to cater for special needs. The framework addresses the following high-level goals:

1. The method of programming the framework for a particular transformation task should encourage, wherever possible, a consideration of the notations themselves rather than the algorithmic methods for their transformation.

2. The framework should encourage the prototyping and customisation of new transformation tasks—this should require as little effort as possible.

3. The framework should integrate with other transformation systems, particularly typesetting systems, so as to take advantage of the enormous amount of research and development that has already been done in this area (see for example Knuth [44]).

4. The framework should further facilitate the creation of new notations on demand in order to address unusual educational difficulties or special circumstances.

This dissertation describes the new framework within the context of existing systems and gives examples that illustrate its potential.

1.6 Structure of dissertation

Chapters 2 and 3 set the work in its historical context by citing examples of related work. Chapter 2 gives some examples of how transformation systems have been used to assist with special needs, while Chapter 3 discusses generalised transformation frameworks that can be programmed for many different applications.

The bulk of the new work is discussed in Chapters 4 through 7. Chapter 4 describes the generalised transformation framework 4DML, while Chapter 5 provides some examples of its use. Chapter 6 evaluates the design of 4DML, and Chapter 7 discusses how it may be implemented efficiently.

Finally, Chapter 8 makes some concluding remarks and discusses possibilities for future work.

2 Special-case transformation systems

2.1 Introduction

This chapter discusses some examples of systems that use transformations to assist people with special needs. The systems are "special case", not because their primary purpose is to help those with special needs—it often is, but there are also *inclusive* systems that aim to cater for others too—but because the transformations they employ are limited to one or two domains (such as Web pages, or mathematics) and, although they might permit some customisation, they cannot easily be programmed to handle completely new transformation tasks. The next chapter discusses *generalised* transformation frameworks, which can be used as programmers' tools to build new systems as needed.

Chapter overview. This chapter begins by discussing assistive technologies for people with limited sight, such as screen enlargers and text-to-speech systems. It then comments on the customisation of on-screen and printed material by using colours and alternative layouts, and outlines the transformations involved in the production of Braille. Finally, it discusses Web mediators, which perform transformations on websites in order to make them more accessible; some discussion of my own mediator is included.

2.2 Assistive technologies for print disabilities

In 1972, Rubinstein and Feldman proposed a Braille terminal for blind computer operators [58]. Computer-driven reading machines for the blind were also conceived at that time [3]. During the 1980s, when MS–DOS was the industry standard, many companies (such as Cobolt Systems, Dolphin Systems, Techno-Vision and PulseData) marketed "adaptive" or "assistive" technologies for blind DOS users. These adaptations usually involved a combination of speech hardware and screen-reading software. Screen magnifiers for partially-sighted people were also common, and some Braille displays were available, although the latter were more expensive and required a knowledge of Braille, which not all blind people had [54].

Problems. Pitt and Edwards [54] evaluated a screen reader and found several inadequacies. Unlike the line-mode terminals of the 1970s, most DOS applications updated the screen in a highly non-linear fashion; simply reading everything as it was written was no longer effective, and algorithms had to be developed to allow blind people to extract the appropriate information from the screen without having to read or listen for too long and without having to remember too much context, since verbal information is transient.

Developing such algorithms often involved making specific allowances for commonly-used applications, usually by writing “profiles” for those applications that included information such as “Don’t bother to read any writes to screen position (1,75) because it’s a clock that’s updated every second”. Trivial changes in the software’s appearance could require the writing of a new profile; this led to a situation where blind people who were not skilled profile writers were effectively restricted to using particular versions of particular pieces of software.

GUIs. The rise of the GUI during the 1990s rendered DOS screenreaders obsolete, and it was some time before effective replacements were developed [69]. As well as the problems that DOS screenreaders had to face, there was now the additional complication that programs can display their controls by writing bitmaps (a signal-based format) instead of using the standard system calls, and interactions can thus be more difficult to intercept. Moreover, the visual concepts of GUIs do not come naturally to computer novices who have been blind from birth.

Research interest in screenreaders included the Mercator project for the X window system [52], and the GUIB project for Windows [75], which even tries to simulate in a speech environment the direct manipulation of positioning windows.

For partially-sighted people, GUIs bring the advantage that it is often possible to customise the size and colour of the text without additional software; such customisation has also been shown to be beneficial to some dyslexics [28], and it is important to note that there is no single configuration that is best for everyone, since the needs involved are diverse. However, many programs do not honour such customisation, or contain display bugs that only manifest themselves when such customisation is in effect. Further, the user interface to the customisation can itself be inaccessible, or unsuitable for novices. Screen magnification software has therefore continued to be marketed.

Navigation. When the amount of data that has to be displayed is significantly greater than what will fit on the display, users have to “navigate” around the data, and can “get lost” (for example, see Watts-Perotti and Woods’ article on the subject [74]). The use of screen magnification, speech, Braille, or any other output method that cannot display as much information simultaneously as the software designer expected, can accentuate this problem.

2.3 Beyond assistive technologies

Because of the above-mentioned problems with assistive technologies, Raman [56, 57], Zajicek et al [81] and others adopted the approach of implementing specialised applications to cater for the needs of blind people, rather than trying to adapt industry-standard systems that were not originally designed for the purpose. Raman extended the EMACS editor with a speech interface that is completely different from its visual interface; he also produced AsTeR, a comprehensive system for reading mathematical documents as speech, either interactively or non-interactively. Zajicek’s work involves a Web browser that provides navigation aids for the blind, involving information-retrieval techniques on complex Web pages.

Increasing the audio bandwidth. As developments in speech synthesis progressed, the speed of the speech could be increased, thus allowing speech users to work faster once they were used to the faster rate. Simply increasing the speed of the speech is not usually as effective as using information retrieval techniques to reduce the amount of text that needs to be read. Calderwood [14] suggested using multiple voices for fast delivery of information in action games. Raman [56] and others convey extra information with background sounds and voice modulations. Some people with limited ability to read a visual display will use the display (perhaps with magnification) in addition to speech.

2.4 Customising Print

Most wordprocessing and similar software allows the print size to be increased, and many programs allow the colours, style of text, and other typesetting parameters to be adjusted. This is helpful to both partially-sighted and dyslexic people [28].

Syntax highlighting. Normally used for programming languages, syntax highlighting involves marking up text with colours (or fonts) to indicate its syntax, as in `for(int i=0; i<10; i++)`. This helps people with print disabilities (provided that they can see the highlighting) because, once they are familiar with the language and its idioms, they can "zoom out" and navigate around the colour pattern without having to read the details. It also assists with fixation—if through physical or other reasons it is difficult to concentrate on a fixed point on the page, then it is often easier to return to that point by using low-resolution colour information.

Colour highlighting has been effectively used in normal text [28] and in music (by *Sibelius* [24]). It is an example of enhancing a notation by adding extra information in a format orthogonal to the original, so as to aid navigation around the notation. It should not be confused with the use of colour for spelling and grammar checking in Microsoft Word, which is not for navigation.

Layout problems. Large text implies that less of it can be fitted into a given area. If the layout is not flexible then it will require such things as laborious horizontal scrolling, or unwieldy large paper that taxes the reprographics facilities.

Even if the layout is flexible, some page-layout algorithms can fail to produce a readable layout when given the extreme constraints that very-large print presents—they have a maximum text-size-to-page-area ratio beyond which they break down. This can be demonstrated by viewing a website with frames and tables on a low-resolution screen at the largest font setting.

For this reason, people who use very large print will often wish to transform tabular layouts into a more flexible form—see Section 2.6.3.

2.5 Braille Typesetting

Algorithms for typesetting Braille text have been developed since the 1980s. Once the Braille codes have been produced, they can be sent to an embosser—a device for producing raised dots on suitable material, sometimes called a "Braille printer"—or to a

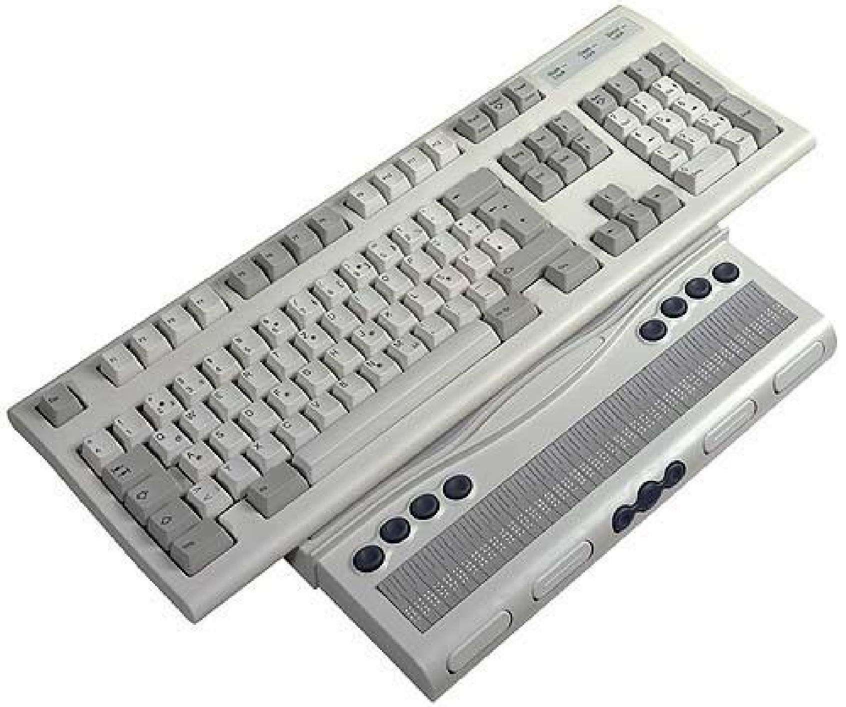


Figure 2.1: A refreshable Braille display attached to a normal keyboard (image by Tieman Group)

refreshable Braille display, which uses mechanical or piezoelectric techniques to raise the dots temporarily (Figure 2.1).

Some of the more well-known typesetting products are the Royal National Institute for the Blind's "Braille-It!", the US National Federation of the Blind's NFBTrans, and the commercial TuxTrans system (which is multilingual and can also translate some mathematics). The state of the art is probably the high-resolution (20dpi) Tiger Advantage embosser with its Windows printer driver that allows almost any document (including diagrams) to be embossed with little fuss, but most establishments have older embossers.

Contractions. The use of "contractions"—abbreviations—can increase the reading speed in alphabetic scripts. This is not just data compression, since many experienced users of Braille associate contractions with phonetic or even semantic concepts—using the contraction for "mother" in the word "chemotherapy" is a bad idea. Even high-end Braille typesetting products are sometimes overzealous in their use of contractions, particularly with newly-invented words like "scandisk" (which contains "and", so is sometimes incorrectly abbreviated "sc&isk"). The use of speech synthesis algorithms could help to avoid cross-syllable contraction.

Braille contractions are not to be confused with the methods used to abbreviate text when sending short messages on mobile telephones (SMS). These abbreviations are usually meant to optimise the time it takes to input the message on the telephone's limited keypad; as a result they often cut across semantic concepts and hence require more mental effort to decipher. Braille contractions are meant to increase the reading speed—this means writing fewer characters for the fingers to pass over, but it also means that the reader should not have to stop and think to decode abbreviations that are applied out of their usual contexts.

A better analogy for Braille contractions is the use of kanji (Chinese characters) in Japanese. It is possible to write Japanese without using kanji—the phonetic scripts (hiragana and katakana) are sufficient to record spoken text (although some written literature is composed in such a way that you cannot interpret it from its sound alone). However, for those who can read it, kanji does increase the reading speed by representing entire concepts in one go. One would only use a kanji that represents a particular word if that precise meaning is intended; it is considered improper to use the "wrong" kanji even if that kanji is a homonym (another word that sounds the same and hence will be read the same as the intended word), because the reader must then apply more thought to deciphering what was meant. Braille contractions are similar, which shows that the cognitive principle at work is not visual association.

Customisability. Most contraction algorithms are rule-based and can be used in another language if the rules are changed appropriately. A problem with many systems is that it is very difficult for the user to customise the rules, which may need to be done in a pedagogical setting. Blind children learn the contractions and word abbreviations of Braille in carefully-graded steps, and while they are doing this they need special texts that use some contractions but not others [25]. If these texts are to be generated automatically then the teacher must be able to customise the list of contractions that can be used, preferably without getting lost in a large database of rules and exceptions in an unfamiliar notation.

East Asian languages. Producing Braille from Chinese and Japanese presents other challenges. There are Braille codes for analytically representing the characters, but most text is more readable when transcribed from one of the phonetic character sets, so the text must first be "read" into sounds [38]. To do this properly requires natural-language processing, because there is usually more than one way of reading any given character, depending on the context. Indeed, the above discussion comparing Braille contractions with kanji does not apply literally—there are far more kanji than can possibly be represented in a reasonable contraction system. English Braille recognises only about 1000 contractions, including those that require more than one Braille cell; the Japanese Standards Association defines over 12,000 kanji with at least 3000 in common use, and in Chinese there are even more characters.

Specialist notations. More general problems are associated with specialist Braille notations such as mathematics, musical scores, chemical bonds, and so on. These notations have many different standards and house styles, and most existing transcription software (such as MFB [45] and Goodfeel [49]) is limited to outputting in very few of them. The problem is further complicated by the fact that the source material is stored in many diverse formats; multiple conversions are often required, sometimes leading to information loss due to the limitations of intermediate formats.

2.6 Transformations for Web Accessibility

The World Wide Web Consortium (W3C) user agent accessibility guidelines [35] epitomise a long history of attempts to address the difficulties faced by special-needs users of the World Wide Web:

> "For instance, flashing content may trigger seizures in people with photosensitive epilepsy... Distracting background images, colours, or sounds may make it impossible for users to see or hear other content... Scripts that cause unanticipated changes may disorient some users"

The W3C recommends that user agents (i.e. browsers) should be customisable to compensate for these and other special needs. Modern Web browsers can be customised with user-supplied CSS (Cascading Style Sheets); a carefully-crafted CSS specification (mine is 246 lines long) can address most accessibility issues. There are exceptions, such as the need to re-organise complex layouts so that the most important information is shown first even on restrictive displays—even with the draft Level 3 standard of CSS, this can be accomplished only in special cases.

Older browsers are often insufficiently customisable to a far greater degree, or they may contain bugs that manifest themselves in unusual configurations and hence make those configurations undesirable. Since a user's choice of software might be restricted by institutional policy, compatibility requirements, and economic circumstances, it may be necessary to transform Web content into a more accessible form before it reaches the browser, to compensate for the browser's limitations.

2.6.1 Accessible authoring

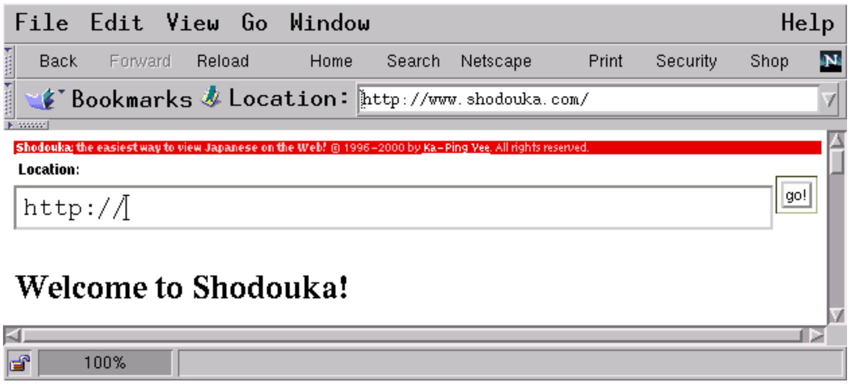
Historically there have been many campaigns to persuade the authors of Web sites to make them more accessible to users with special needs, with particular emphasis on totally-blind users. The formation of the W3C's Web Accessibility Initiative (WAI) in 1998 unified many of these, and acts of law (such as the USA's Americans with Disabilities Act 1990 and the UK's Disability Discrimination Act 1995) tried to enforce Web accessibility to prevent social exclusion.

These aims were difficult to uphold universally, as most Web authors were unaware of the guidelines, or considered them to be too time-consuming to implement, or overly restrictive on their artistic freedom. Moreover, many produced Web pages without knowledge of the underlying technicalities, using application software that did not conform to the WAI's recommendations.

Even if all websites could be made "accessible", this would not eliminate the need for customisability, because different special needs give different requirements. Authors can aid customisability, for example by avoiding dependence on signal-based formats such as images (providing textual alternatives where meaningful), but ultimately the browser must be sufficiently customisable, and if it is not then other software is needed to compensate.

2.6.2 Mediators

A mediator is a server that retrieves pages from other servers on behalf of a client, like a proxy, but which performs some useful transformations on the pages before returning


File Edit View Go Window Help
Back Forward Reload Home Search Netscape Print Security Shop
Bookmarks Location:
go!
100%

http://www.shodouka.com/

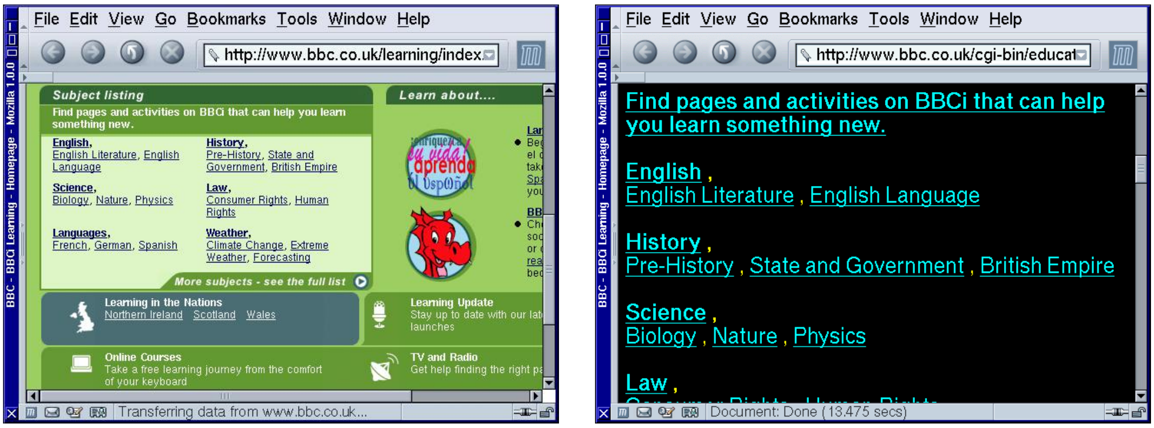
Shodouka: the easiest way to view Japanese on the Web! @ 1996-2000 by Ka-Ping Yee. All rights reserved.

Location:

http://

Welcome to Shodouka!

Figure 2.2: Shodouka entry form in a Web browser


BBC - BBCi Learning - Homepage - Mozilla 1.0.0
File Edit View Go Bookmarks Tools Window Help
Subject listing
Learn about....
More subjects - see the full list
Learning in the Nations
Learning Update
Online Courses
TV and Radio

http://www.bbc.co.uk/learning/index

Find pages and activities on BBCi that can help you learn something new.

English, English Literature, English Language

History, Pre-History, State and Government, British Empire

Science, Biology, Nature, Physics

Law, Consumer Rights, Human Rights

Languages, French, German, Spanish

Weather, Climate Change, Extreme Weather, Forecasting

Northern Ireland Scotland Wales

Stay up to date with our lat launches

Take a free learning journey from the comfort of your keyboard

Get help finding the right pa

Transferring data from www.bbc.co.uk...

(a) Before: Original page

http://www.bbc.co.uk/cgi-bin/educat

Find pages and activities on BBCi that can help you learn something new.

English ,
English Literature , English Language

History ,
Pre-History , State and Government , British Empire

Science ,
Biology , Nature , Physics

Law ,

Document: Done (13.475 secs)

(b) After: page processed by BETSIE

Figure 2.3: The effect of a mediator: BETSIE on the BBC's website

them to the client [79]. This has the advantage of being independent of both the client software and the other servers.

- The first mediator was probably Shodouka [80], a service for displaying Japanese Web pages on browsers that do not have Japanese fonts, by using graphics to show the characters. Entry to Shodouka is by means of an HTML form (Figure 2.2), and when a page is retrieved, all the links on it are modified to point back through the mediator, so links can be followed without leaving the mediator. Many later mediators also used this concept.

- WAB [39] is an HTTP proxy (based on CERN httpd) that modifies HTML to assist users who are totally blind. It was not customisable, and its being a proxy meant that it could not be used by individuals who were only allowed to use their institution's proxy, unless they could have a local copy installed.

- BETSIE [51] is a simple Perl script written by Wayne Myers for the British Broadcasting Corporation (BBC) in conjunction with the UK's Royal National Institute for the Blind (RNIB). It was originally intended to make the BBC's website more accessible—see Figure 2.3—but it is also available for download. The intention is that a webmaster of an inaccessible site can install BETSIE, possibly adding code that is specific to the site (as is done by the BBC). It is therefore assumed that there is some co-operation between the mediator and the webmaster, and thus the mediator does not have to handle every possible HTML technique. The BBC's website is large and rapidly changing, with many webmasters and a large amount of dynamically-generated HTML, so it is difficult to check that BETSIE always works.

Some websites have separate "text only" versions. These can be useful, but due to lack of maintenance they are often out-of-date or incomplete, so people who could benefit from them can be reluctant to use them. Using a system such as BETSIE to automatically generate the text-only pages would be an improvement on this. However, since BETSIE can perform much less adequately than manual efforts (for example, see Figures 5.1 and 5.2 on pages 78 and 79), it would be better to use a transformation system such as XML publishing to generate the different versions of the pages from higher-level data. In the case of the BBC this might be prohibited by the size and structure of the corporation, since the website is managed by many different departments.

- The concept of using knowledge of specific websites to enhance the processing of those sites is also used in Asakawa and Takagi's proxy for blind users [4, 64], which can use volunteer-supplied knowledge of a site's house style to determine the order in which the sections of each page should be presented. Although this knowledge does have to be acquired and is liable to be obsoleted by site redesign, the proxy can in some cases acquire it automatically by using heuristics on a single page, or by comparing different pages of the site or different versions of a page. This latter technique is similar to that used by Ebina, Igi and Miyake [18] in their extraction of updated content. Since site-specific knowledge takes time to acquire, it is more appropriate if the mediator has a large user base and the site is popular, which is not always the case.

  Asakawa and Takagi's proxy can be customised; each user's customisation is stored on the server, rather than being encoded in the page's links. Proxy authentication is used to identify each user. Where users do not have suitable client software or cannot change their proxy settings, a second mediator can be used, which encodes the user's identification in the page's links and mediates between the client and the proxy. The proxy itself is based on IBM's WebSphere Transcoding Publisher [9], which is normally used for e-business and middleware; this makes for a robust implementation. IBM has since developed the idea in its WBI (Web Intermediaries) Development Kit and the Aurora project [31].

- A popular mediator is Muffin [22], which is intended for use as a personal proxy. Muffin can be customised only by the proxy's administrator, but the intention is that each user administrates his or her own copy. This is a client-side approach and is therefore not always possible if the user's choice of software is restricted.

- A little-known mediator for users with low vision is AlterPage [1], intended for users of the 'WebTV' set-top box service in the USA. AlterPage has limited capabilities, but some WebTV users find its use of WebTV-specific markup to be helpful (Petro Giannakopoulos, private communication).

The World Wide Web Consortium maintains a list of evaluation and repair tools [15] that includes several other mediators.

2.6.3 The Access Gateway

The Web Access Gateway [13] is a mediator that was designed with my personal experience as an individual with low vision in mind. In some cases it is also usable by people with other difficulties such as dyslexia. It was intended to allow people with low vision to access *any* Web page that is usable by fully-sighted people, without needing the co-operation of webmasters or volunteers.

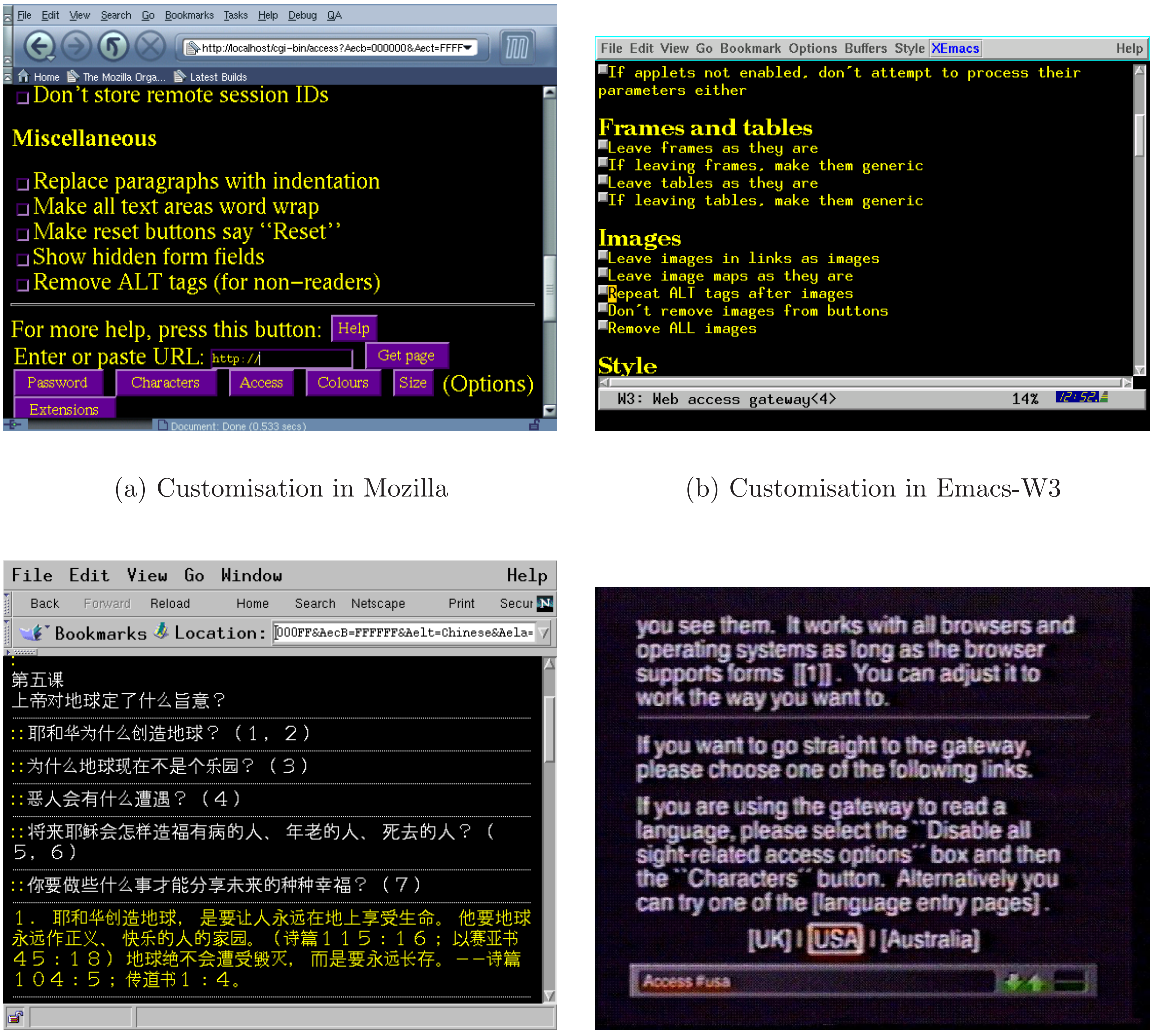
The Access Gateway has been implemented as a CGI script in C++. The URL of the page to be processed, and any preferences, are submitted to the mediator through an HTML form, and the response is modified so that all links and forms point back through the mediator and specify the same settings.

Completeness. The gateway was designed to handle frames, tables, images and maps with missing or meaningless ALT attributes, lists of adjacent links, forms, authentication, SSL, Java, JavaScript, cookies, Flash, plug-ins, Cascading Style Sheets, HTML errors, referer tracking, browser checks, automatic refresh, cluttered layouts, ASCII art, surplus spaces between characters, formatting directives that depend on the display dimensions, and any of the world's commonly-used character encodings. As shown in Figure 2.4, it is usable in any Web browser on any operating system, without having to install additional software or change any of the browser's settings, and it is usable over low bandwidth. Many browser bugs are worked around as necessary.

It is often necessary to display URLs to the user, either in the status line of certain browsers as the pointer passes over a link, or because no other textual information about the link is available. The gateway can present these URLs in an abbreviated form so that the reader can extract meaningful information more easily.

Customisability. The program is customisable by individual users (Figure 2.4a, 2.4b), and can support many sessions simultaneously, each with its own settings. The customisation interface itself honours the user's preferences, responding to changes immediately where possible. The user's customisation is preserved between pages, so that users can visit other pages (such as by following links) without having to re-enter their customisation. However, the mediator is stateless—the settings are stored by the client rather than the server, and so are not subject to space restrictions or timeouts. Users can pass their settings on to others, publish them as presets, use them on different installations of the program, store them persistently in bookmarks, or create alternative interfaces to the gateway, as some organisations have done.

Information loss and side-effects. Attempting to make pages accessible by removing unwanted HTML markup is not sufficient. For example, removing table markup can leave unterminated links and unnecessary "spacer" objects. The Access Gateway makes an effort


File Edit View Search Go Bookmarks Tasks Help Debug QA
http://localhost/cgi-bin/access?Aecb=000000&Aect=FFFF
Home The Mozilla Orga... Latest Builds

□ Don’t store remote session IDs

Miscellaneous

□ Replace paragraphs with indentation
□ Make all text areas word wrap
□ Make reset buttons say “Reset”
□ Show hidden form fields
□ Remove ALT tags (for non–readers)

For more help, press this button: Help
Enter or paste URL: http:// Get page
Password Characters Access Colours Size (Options)
Extensions

Document: Done (0.533 secs)

(a) Customisation in Mozilla

File Edit View Go Bookmark Options Buffers Style XEmacs Help

□ If applets not enabled, don’t attempt to process their parameters either

Frames and tables

□ Leave frames as they are
□ If leaving frames, make them generic
□ Leave tables as they are
□ If leaving tables, make them generic

Images

□ Leave images in links as images
□ Leave image maps as they are
□ Repeat ALT tags after images
□ Don’t remove images from buttons
□ Remove ALL images

Style

(b) Customisation in Emacs-W3

File Edit View Go Window Help
Back Forward Reload Home Search Netscape Print Secur
Bookmarks Location: 000FF&AecB=FFFFFF&Aelt=Chinese&Aela=

第五课
上帝对地球定了什么旨意？

::耶和华为什么创造地球？（１，２）

::为什么地球现在不是个乐园？（３）

::恶人会有什么遭遇？（４）

::将来耶稣会怎样造福有病的人、年老的人、死去的人？（５，６）

::你要做些什么事才能分享未来的种种幸福？（７）

１．耶和华创造地球，是要让人永远在地上享受生命。他要地球永远作正义、快乐的人的家园。（诗篇１１５：１６；以赛亚书４５：１８）地球绝不会遭受毁灭，而是要永远长存。——诗篇１０４：５；传道书１：４。

(c) In use on a Chinese page

you see them. It works with all browsers and operating systems as long as the browser supports forms [[1]]. You can adjust it to work the way you want to.

If you want to go straight to the gateway, please choose one of the following links.

If you are using the gateway to read a language, please select the ``Disable all sight-related access options'' box and then the ``Characters'' button. Alternatively you can try one of the [language entry pages].

[UK] | [USA] | [Australia]

(d) In use on WebTV set-top box

Figure 2.4: Screen shots of Web Access Gateway

to prevent these unwanted side-effects. It is also able to compensate for the information loss that might occur when markup is removed. For example, if a page author uses colour or layout to convey information, then that information might be lost when the display parameters are changed to the user's preferences. The mediator can convey some of this information in another way, but without unnecessarily cluttering the output, by inserting various punctuation characters into the text, or using multiple colours but selecting them from a user-supplied list.

Server requirements. The implementation is portable across different server platforms, and is believed to be secure against remote exploits and denial of service attacks, but without unnecessarily compromising functionality. It is hosted on servers that do not insert advertisements or other clutter. Some establishments require access to the program to be restricted to internal clients, and some have disabled certain features to reduce traffic. Some of the installations of the program are aware of each other, so that they offer to re-direct users to a closer mirror if it is sufficiently capable.

Extensibility. Since trends in Web technologies change over time, the implementation is extensible with new features as the need arises.

Cookies. Since cookies are part of the user's state, the Access Gateway treats them in the same way as user preferences, so sites that require cookies work even if the browser does not support them. However, the need to encode the state (including the cookies) in link URLs limits the total size of stored cookies. Further, encoding data in link URLs significantly increases the size of a page containing many links, which can cause problems when the client has low bandwidth. Web Access Gateway compresses its URLs by abbreviating certain combinations of options, although further compression is conceivable. It can also make use of client-side cookies if the client supports them, but this is not required.

Security. It is difficult to mediate secure sites without compromising security. An encrypted connection can be made between the mediator and the remote site, and possibly between the client and the mediator, but the mediator itself must be trusted. Web Access Gateway contains no cryptographic code, in order to remain legal if strong cryptography is banned. However, it can run on a secure server and it can use the SSL version of Lynx (if available) to fetch pages. Basic authentication, being insecure, is trivial to mediate.

Language viewer. The Access Gateway can read a number of character encodings and detect which one to use based on frequency tables. The most popular use of Access Gateway is as a character-set converter that uses images for characters that are not supported by the client; this works best in Chinese, Japanese and Korean, and the disability-related functions are normally switched off. One installation of the Access Gateway, dubbed "Monash University ACCESS–J Japanese Web Page Viewer", is cited by numerous organisations as a recommended resource for browsing Japanese pages on systems that cannot otherwise display them adequately.

Heuristics. The page's reading order is difficult to determine automatically; the Access Gateway tries to output meaningful content first by using a simple heuristic based on link density. Further work is needed in this area.

Dealing with images that do not have alternative text (ALT attributes), or that have unhelpful ALT attributes like "image", requires heuristics; it is especially important to provide text for navigational links. Some mediators (e.g. Asakawa and Takagi's proxy) substitute the title of the page that is being linked to; Access Gateway tries to extract text from the URL, since this avoids the need for fetching further pages during processing. A problem is that both URLs and titles can be meaningless. Dardailler [17] proposes a repository of ALT attributes for frequently-used websites; however, access to other sites can still be important.

2.6.4 Conclusion

Mediators can do much to alleviate the problems with Web pages that are experienced by users with special needs, but it is very difficult for mediators to eliminate these problems completely.

The Access Gateway fulfils most of its objectives, although it still has problems transforming some websites, particularly sites involving online purchasing. At the time of writing, such sites are usually tested only on Microsoft Internet Explorer, and it is risky to try them with other browsing software because it could expose bugs that lead to incorrect billing.

Mediators are coming into mainstream use for Web-enabled mobile telephones, where the client's processing and bandwidth is limited; both WAP and iMode make use of mediators. The small size of mobile displays produces layout problems similar to those associated with large print on a desktop screen, so the mobile market might generate more interest in researching this area.

2.7 Summary

The systems discussed in this chapter provide examples of transformations that assist people with special needs, although some of them (such as Muffin, page 23) are not primarily for this purpose. The chapter included assistive technologies such as text-to-speech software, customised print such as syntax highlighting, Braille typesetting, and increasing the accessibility of the World Wide Web. These systems are all "special case" in that their transformations are limited in scope; the next chapter describes more generalised transformation frameworks.

The Web Access Gateway is a Web mediator designed to assist people with low vision and related difficulties to access websites independently without needing special software—the mediator transforms the page before it reaches the client computer. The primary contribution of this project is to show the many issues that needed to be addressed in a production system suitable for general-purpose Web browsing, since previously-existing solutions were more limited.

The Access Gateway is in regular use by several hundred people. Institutions that have installed it on their servers include 3 US government agencies, 2 specialist Internet service providers and 5 non-profit organisations; there may be others that are not known to the author, since the software is freely available.

3 Generalised transformation frameworks

3.1 Introduction

This chapter reviews some transformation systems that are programmable—they can be used as a basis for implementing new transformation tasks as needed. These systems are not limited to one domain, but deal with generalities, such as symbols and data, that can apply to many domains. A generalised system that is used in the field of software engineering, for example, might equally well be used to accomplish transformation tasks when dealing with mathematical or musical notation.

Chapter overview. After some overall remarks on programmable systems, this chapter discusses Unix tools that are related to transformation, as well as parser generators and re-writing systems such as TXL (page 39). It then introduces XML-based systems and comments on the implications of matrix-like data structures and multiple parse trees, which are frequently the more difficult cases both in XML and in the other frameworks discussed in this chapter.

Differentiated from general-purpose programming. The notion of transforming data from one structure to another is a very general one. Nearly every program ever written can be thought of as a transformer, interpreting its input (perhaps a sequence of commands from the user) and producing some output (perhaps feedback to the user as the input is being given). Conversely, a given transformation could conceivably be implemented in virtually any programming language.

As is the case with many programming problems, however, the class of transformations that form the main focus of this thesis—the conversion and adaptation of the notations of various educational disciplines—can be achieved more easily with some programming tools and languages than with others. While different programmers have different ideas of what is easy and what is difficult, it is still possible to achieve some generality by stating that, if a tool or language was developed specifically to support a particular design approach that is well-suited to a certain class of problems, then many people who wish to solve problems in that class are likely to find that tool or language easier to learn and use than a more general programming language, and the associated design approach is more likely to be taken, hence reducing the amount of work that might otherwise result.

Many transformation tools are actually Turing-powerful and could potentially be used as general-purpose programming languages, but it is still useful to make the distinction, because these tools are *biased* toward a particular class of problems and a particular design approach; if they are used outside that class, the resulting code may not seem so high-level, particularly if there is overt *emulation* of the behaviour of other general-purpose programming systems.

Trade-off. Any programmable system must strike a balance between versatility and simplicity. The limit of versatility is to require the user to write the entire program in low-level code, but that gives little simplicity; the limit of simplicity is to forbid any programming (just provide built-in special-case functionality), but that gives little versatility. Between these limits lies a complex non-linear relationship that depends on design decisions. These should make the system powerful enough to serve its purpose, but simple enough to justify its use as a tool.

3.2 Unix tools

Many command-line tools commonly associated with the Unix environment are suitable for performing transformation operations on text-based data. Bentley [10] describes the task-specific languages of these tools as "little languages"; several are reviewed in Salus [59]. Three themes often recur: regular expressions, pipes, and preprocessors.

3.2.1 Regular expressions

The non-interactive stream editor, `sed`, can perform regular-expression based substitution operations on its input, and has a command language similar to that of the interactive editors `ed` and `vi`. This language is Turing-powerful—a Turing machine can be emulated by a set of search-and-replace commands in a loop—and `sed` scripts exist for numerous transformation tasks.

Incorporation into other languages. Since `sed` scripts are cryptic and can be difficult to maintain, the support for regular expressions and other text processing has also become a major feature of some scripting languages with higher-level constructs, such as Awk, Perl, Ruby, Python, and Emacs lisp; these languages are frequently used to achieve transformations.

Comparison with SNOBOL. From 1962 to 1967, Bell Labs developed the String-Oriented Symbolic Language, or SNOBOL [29]. This utilises a language for pattern matching which is considered by some to be clearer than regular expressions as well as being more powerful—it supports recursively-defined patterns and is in some respects like a parser generator (Section 3.3) with full backtracking. At present it is rarely used due to its relative slowness and the consequent prevalence of regular-expression based systems. In comparison with regular expressions, SNOBOL code tends to use fewer special characters and a higher-level structure—compare Figure 3.1 with Figure 3.2.

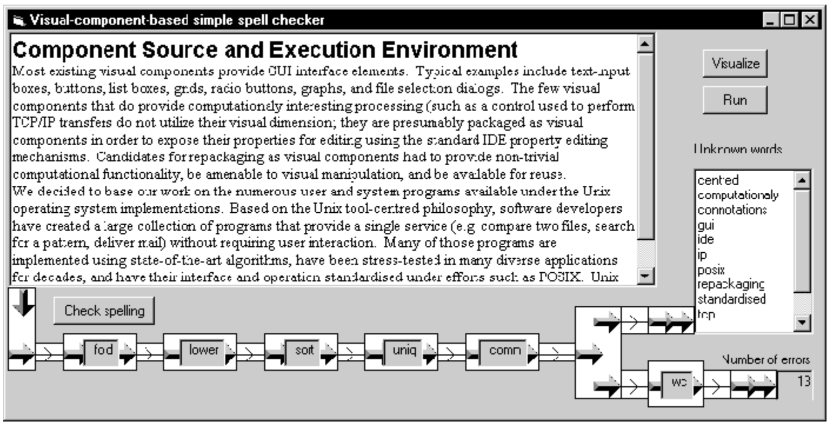
Example. Figure 3.1 shows a `sed` script to convert simple arithmetic expressions in standard infix notation into Lisp's prefix notation; thus `1*(2+3)-4` would become

```
#!/bin/sed -f

:loop

# On later iterations of this loop, innermost brackets will
# have been replaced with l and r

# remove unneeded brackets
s/(\(l[^r]*r\))/\1/g

# process multiplication/division
s/\(l[^r]*r\|[0-9]\+\)\([*/]\)\(l[^r]*r\|[0-9]\+\)/(\2 \1 \3)/g

# expand out to next level of brackets
s/(\([^()]*\))/L\1R/g
s/l/</g
s/r/>/g
s/L/l/g
s/R/r/g

# repeat until no more changes
t loop

# recognise innermost brackets, replace with l & r
s/(\([^()]*\))/l\1r/g
t loop

# now restore the brackets
s/</(/g
s/>/)/g

# now do all that again but with addition/subtraction
:loop2
s/(\(l[^r]*r\))/\1/g
s/\(l[^r]*r\|[0-9]\+\)\([+-]\)\(l[^r]*r\|[0-9]\+\)/(\2 \1 \3)/g
s/(\([^()]*\))/L\1R/g
s/l/</g
s/r/>/g
s/L/l/g
s/R/r/g
t loop2
s/(\([^()]*\))/l\1r/g
t loop2
s/</(/g
s/>/)/g
```

Figure 3.1: Infix to postfix in `sed`

```
      Fac = SPAN('0123456789') | '(' BAL ')'

      E = INPUT

loop1 E  Fac . f1  ANY('*/') . op  Fac . f2  =  '(' op ' ' f1 ' ' f2 ')' :S(loop1)
loop2 E  Fac . f1  ANY('+-') . op  Fac . f2  =  '(' op ' ' f1 ' ' f2 ')' :S(loop2)
loop3 E  '(' Fac . f ')' = f :S(loop3)

      OUTPUT = E
END
```

Figure 3.2: Infix to postfix in SNOBOL4



Component Source and Execution Environment

Most existing visual components provide GUI interface elements. Typical examples include text-input boxes, buttons, list boxes, grids, radio buttons, graphs, and file selection dialogs. The few visual components that do provide computationaly interesting processing (such as a control used to perform TCP/IP transfers do not utilize their visual dimension; they are presumably packaged as visual components in order to expose their properties for editing using the standard IDE property editing mechanisms. Candidates for repackaging as visual components had to provide non-trivial computational functionality, be amenable to visual manipulation, and be avalable for reuse.

We decided to base our work on the numerous user and system programs available under the Unix operating system implementations. Based on the Unix tool-centred philosophy, software developers have created a large collection of programs that provide a single service (e.g compare two files, search for a pattern, deliver mail) without requiring user interaction. Many of those programs are implemented using state-of-the-art algorithms, have been stress-tested in many diverse applications for decades, and have their interface and operation standardised under efforts such as POSIX. Unix

Unknown words

centred
computationaly
connotations
gui
ide
ip
posix
repackaging
standardised
tcp

Number of errors
13

Figure 3.3: Constructing pipelines graphically—image taken from Spinellis' paper [60]

(- (* 1 (+ 2 3)) 4). Where appropriate, this transformation is also used in the other examples of this chapter to illustrate different transformation systems.

The sed script is cryptic, making maintenance difficult—consider for example the task of adapting it to support the use of '-' as a negation operator, to accommodate expressions such as 3*-(4+5). Although this can be done, it requires some thought. The SNOBOL version (Figure 3.2) is more readily adaptable.

Note: Figure 3.2 utilises SNOBOL's BAL primitive to match balanced parentheses. If BAL were not present then it can be emulated briefly enough with a recursive pattern:

```
BAL = ARBNO(NOTANY("()") | "(" *BAL ")")
```

3.2.2 Pipes

Unix shells, such as sh, bash, zsh, ksh, csh, ash, tcsh etc, support various control-flow features and allow the construction of pipelines to pass the output of one command to the input of another, perhaps with other commands acting as "filters" to perform transformations on data while it is in the pipeline. Hence complex transformations can be built up from smaller primitives. The Unix environment provides many commands that can be used in pipelines to obtain such functionality as sorting, searching, and arithmetic evaluation. It is also possible to construct pipelines graphically—see for example Spinellis [60] (Figure 3.3).

3.2.3 Preprocessors and macros

A preprocessor copies its input to its output, checking for certain embedded codes and acting on them as it does so. Generally the embedded code that the preprocessor recognises is interpreted, executed, and replaced with its output. This is termed "preprocessing" because the output produced is frequently passed to another program, such as a compiler or a typesetting package, for further processing and/or display.

Two modes of operation. When considering a preprocessor's role in the conversion of notations, it is useful to differentiate between the input of the *transformation* and the input of the *preprocessor*. Preprocessors can be thought of as having two distinct modes of operation:

1. The input of the preprocessor is fixed, and encodes the nature of the transformation. The input of the transformation is provided separately, in the form of the preprocessor's configuration or environment, and is queried by the embedded code.
2. The input of the preprocessor is the input of the transformation. The nature of the transformation is governed by the configuration of the preprocessor or by including a library of code and definitions.

Fixed input. The first mode is used when the overall form of the desired output is fixed, but a limited amount of it changes with the input. Input-dependent code is embedded into a fixed output document. This is the approach taken by server-side embedded scripting languages on Web servers, and by the C preprocessor when it is used to customise some code according to the user's desired configuration.

A well-known example of server-side scripting is PHP; server-side scripting systems are also available for several existing programming languages including Python, Java, Tcl, Perl and Scheme, as well as a few specially-designed languages.

User-supplied input. The second mode is used to accomplish more complex tasks. The macro language built in to the typesetting software TEX [44], for example, has enabled other languages to be transformed into plain TEX for typesetting; this includes the LATEX language for structured documents, MusiXTEX [66] for musical notation, and XML formatting objects via PassiveTEX [55]. Sometimes the macros redefine the input language completely, but usually the two languages can be mixed and the preprocessor effectively adds new features to the output language.

m4. The popular macro processor `m4`, derived from GPM [63], is Turing-powerful and has been used in both of the above-described modes of operation. `m4` was originally used to implement Ratfor [40], a preprocessor of FORTRAN that allows C-like control structures; nowadays it is most frequently used in source code distribution systems such as GNU `autoconf`.

3.3 Parser generators

A parser is a program that takes a stream of input and a grammar, and uses the grammar to calculate the parse tree (the hierarchical structure) of the input. Actions are taken on the resulting structure. This can be used as the basis of a transformation. Since much of the effort of creating a parser can be automated by using a parser generator, or "compiler compiler" [2], this method can be useful when implementing some transformations, particularly those that follow a complex input structure. Attribute grammars [42] are frequently used—working up from the bottom of the parse tree, terminals are used to calculate the attributes of non-terminals, which in turn influence the attributes of higher non-terminals until it is possible to take action on a non-terminal and its attributes. It is still necessary to write the "action" code in a general-purpose programming language that is supported by the parser generator, but for some transformations this might be as simple as printing out the parsed data in a different order.

Yacc and related tools. Perhaps the most famous example of a parser generator is the Unix tool Yacc (Yet Another Compiler Compiler) and its GNU equivalent, Bison; this requires auxiliary code to *tokenise* the input, i.e. to group the characters into units such as integers and identifiers. Normally the associated tool `lex`, which is based on regular expressions, is used as a tokeniser. These tools are designed primarily for speed—creating fast parsers is desirable when they will be used in production compilers, as these may be given large quantities of code. However, in the context of transforming notations, particularly for people with special needs, the speed of the parser might be less important than the usability of the parser generator—if there is less scope for error when writing the grammar for the parser generator, then transformations can be prototyped more quickly and with less effort and resources; in many cases this will outweigh the longer runtimes of less-efficient parsers.

Example. Figure 3.4 shows the Yacc version of the infix-to-postfix transformation introduced earlier. Minimal supporting C code is included; the parsing of numbers would normally be done by the lexer. The use of C makes it necessary to write memory-management code; development can be faster when using a scripting language that supports automatic memory management. Figure 3.5 shows the same transformation in Python using the PLY (Python Lex-Yacc) library [7].

Grammar specification. Parser generators allow the grammar of the input language to be specified as a set of rules that determine possible ways to make up the input. The grammar definition language is usually based on Backus-Naur Form (BNF) [43]. This can be more difficult to write than might at first appear, due to the limitations of parser generators:

- Most parser generators are LALR(1)—they have only one token of lookahead—so the programmer must modify the grammar to conform to LALR(1), i.e. to be deterministic in every part of the input;
- Some parser generators do not support the specification of mathematical precedence and associativity, meaning that the user has to specify this in the grammar; this

```
%{
#include <stdio.h>
#include <ctype.h>
#include <malloc.h>
  char* concat(char* a,char* b,char* c,char* d,char* e);
#define YYSTYPE char*
%}

%%

start: expression { puts($$); }; /* output the result */

expression: term { $$ = $1; }
| expression ’+’ term { $$ = concat("(+ ",$1," ",$3,")"); }
| expression ’-’ term { $$ = concat("(- ",$1," ",$3,")"); };

term: factor { $$ = $1; }
| term ’*’ factor { $$ = concat("(* ",$1," ",$3,")"); }
| term ’/’ factor { $$ = concat("(/ ",$1," ",$3,")"); };

factor: num { $$ = $1; } | ’(’ expression ’)’ { $$ = $2; };

num: digit { $$ = $1; }
| digit num { $$ = concat($1,$2,"","",""); };

digit: ’0’ { $$ = "0"; } | ’1’ { $$ = "1"; }
| ’2’ { $$ = "2"; } | ’3’ { $$ = "3"; } | ’4’ { $$ = "4"; }
| ’5’ { $$ = "5"; } | ’6’ { $$ = "6"; } | ’7’ { $$ = "7"; }
| ’8’ { $$ = "8"; } | ’9’ { $$ = "9"; };

%%

yylex () {
  int c=’ ’; while(isspace(c)) c=getchar();
  if (c==EOF) return 0; else return c;
}

char* concat(char* a,char* b,char* c,char* d,char* e) {
  char* scratch=malloc(1000);
  sprintf(scratch,"%s%s%s%s%s",a,b,c,d,e);
  return scratch;
}

yyerror (char* s) { fprintf(stderr,"%s\n",s); }
main() { yyparse(); }
```

Note: For brevity, `concat()` has a poor implementation—it has a fixed number of arguments, is subject to buffer overruns and has a memory leak. Production code should be more complex.

Figure 3.4: Infix to postfix in Yacc

```
import lex,yacc

tokens = ( ’num’, ’plus’, ’times’, ’lParen’, ’rParen’)
t_ignore = " \t\n"
t_num    = r’[0-9]+’
t_plus   = r’[+-]’
t_times  = r’[*/]’
t_lParen = r’\(’
t_rParen = r’\)’

def t_error(t): raise t
lex.lex()

def p_start(t):
    ’start : expression’
    print t[1]
def p_expr1(t):
    ’expression : term’
    t[0] = t[1]
def p_expr2(t):
    ’expression : expression plus term’
    t[0] = "(" + t[2] + " " + t[1] + " " + t[3] + ")"
def p_term1(t):
    ’term : factor’
    t[0] = t[1]
def p_term2(t):
    ’term : term times factor’
    t[0] = "(" + t[2] + " " + t[1] + " " + t[3] + ")"
def p_factor1(t):
    ’factor : num’
    t[0] = t[1]
def p_factor2(t):
    ’factor : lParen expression rParen’
    t[0] = t[2]

def p_error(t): raise t
yacc.yacc()
yacc.parse(raw_input())
```

Figure 3.5: Infix to postfix in Python using PLY

can be difficult for novices—Figure 3.6 looks very much like Figure 3.5, but contains a bug (the associativity of subtraction is incorrect—$1 - 2 - 3$ would be interpreted as $1 - (2 - 3)$ rather than $(1 - 2) - 3$).

GLR (generalised look-right) parsers, such as Elkhound [50], PRECC [12] and more recent versions of GNU Bison, help to relieve the problem by supporting arbitrary grammars, using backtracking if necessary. Even though this may be less efficient, it easier to prototype for three reasons:

1. The user rarely needs to address the ambiguity of the grammar.
2. A separate tokeniser is not required; tokens can be included literally in the grammar, hence reducing the level of skill that is required of the parser generator's user.
3. Actions can be "delayed". Normally actions are included only at points where enough tokens have been parsed to ensure that the parser is taking the correct path. Backtracking parsers execute actions retrospectively once the parse is known, so it is possible to include them anywhere. Figure 3.7 illustrates this with the infix-to-postfix transformation—because it is possible to include output instructions at the *beginning* of the reduction rules, the need to pass data structures up the tree can be avoided. This would not work in Yacc.

However, there is a problem with Figure 3.7—it contains the same error as Figure 3.6; the associativity of subtraction is incorrect. Correcting the error will cause PRECC to enter an infinite loop, since it is a recursive-descent parser and cannot accept rules of the form $E \rightarrow E \ldots$. Figure 3.8 shows an attempt to correct the grammar and to re-write it in a form that Bison and `btyacc` will accept—these tools use "bottom up" shift/reduce parsing with backtracking, so they do not have the same limitations as recursive descent. However, this too is flawed—an infinite loop is still caused, this time by the parser's attempt to explore the infinite series of ε-reductions that is associated with an action at the beginning of a left-recursive rule. This demonstrates that contemporary parser generators, while helpful, do require the user to have some understanding of the theory, and not just the ability to write Backus-Naur Form grammars with arbitrary actions.

Development environment. Some commercial tools such as ProGrammar [68] allow grammar specifications to be constructed and tested graphically; here there is no semantic difference in the workings of the grammar specification—it is simply presented in a different form (Figure 3.9).

3.4 Rewriting systems

Rewriting theory is often used in equational reasoning, including automated deduction, automated verification of specifications, type theory, and so on. It is sometimes called "rule-based programming". A rewriting system involves a collection of "rewrite rules", which are directed equations; informally, they say "whenever you see this, rewrite it as that". These rules are applied in sequence in accordance with a *rewriting strategy*, which is usually a *normalising* strategy, meaning that the structure is repeatedly transformed

```
import lex,yacc

tokens = ( ’num’, ’plus’, ’times’, ’lParen’, ’rParen’)
t_ignore = " \t\n"
t_num    = r’[0-9]+’
t_plus   = r’[+-]’
t_times  = r’[*/]’
t_lParen = r’\(’
t_rParen = r’\)’

def t_error(t): raise t
lex.lex()

def p_start(t):
    ’start : expression’
    print t[1]
def p_expr1(t):
    ’expression : term’
    t[0] = t[1]
def p_expr2(t):
    ’expression : term plus expression’
    t[0] = "(" + t[2] + " " + t[1] + " " + t[3] + ")"
def p_term1(t):
    ’term : factor’
    t[0] = t[1]
def p_term2(t):
    ’term : factor times term’
    t[0] = "(" + t[2] + " " + t[1] + " " + t[3] + ")"
def p_factor1(t):
    ’factor : num’
    t[0] = t[1]
def p_factor2(t):
    ’factor : lParen expression rParen’
    t[0] = t[2]

def p_error(t): raise t
yacc.yacc()
yacc.parse(raw_input())
```

Figure 3.6: An incorrect version of Figure 3.5

```
#include "cc.h"

@ expression = {: printf("(+ "); :} term <’+’> expr2
@            | {: printf("(- "); :} term <’-’> expr2
@            | term

@ sp_expr = {: printf(" "); :} expression {: printf(")"); :}

@ term = {: printf("(* "); :} factor <’*’> term2
@      | {: printf("(/ "); :} factor <’/’> term2
@      | factor

@ term2 = {: printf(" "); :} term {: printf(")"); :}

@ factor = <’(’> expression <’)’> | num

@ num = {digit}+

@ digit = (isdigit)\x {: putchar($x); :}

MAIN(expression)
```

Figure 3.7: Infix to postfix in PRECC (incorrect)

until it is in “normal” form and the rules cannot effect further changes. Rewriting is Turing-powerful.

Differentiated from search and replace. The difference between rewriting and simple “search and replace” (with or without regular expressions) is that rewriting operates on hierarchical structures, rather than on an unstructured string of characters. A rewriting system will operate with data that has already been parsed. Hence it is possible to create rules that specify such things as what *types* of data the rule applies to and in what context, possibly with other conditions added; rule patterns frequently correspond to fragments of the parse tree. The resulting rules tend to be more concise than their search-and-replace counterparts. Rewriting strategies sometimes specify that the pattern replacement must first occur on a particular part of the parse tree, such as the outermost level.

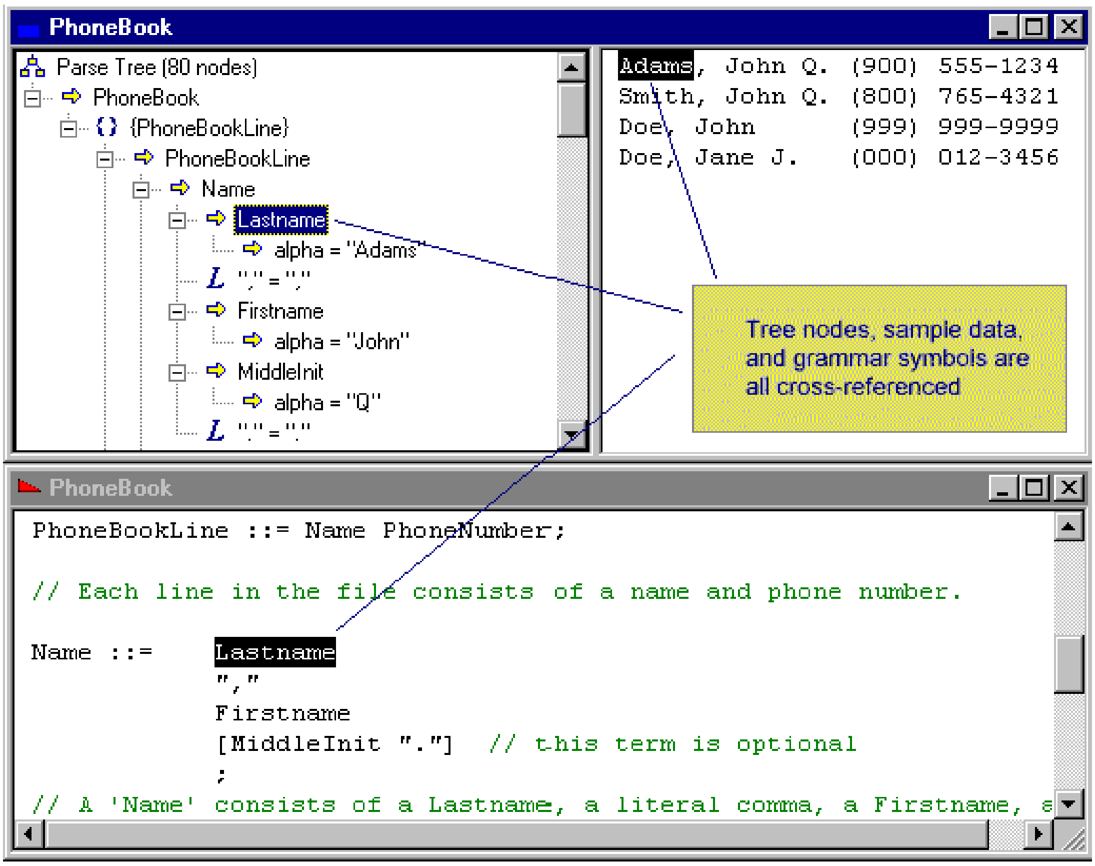
Examples. TXL [16] is a generic rewriting language that incorporates a parser generator, allowing arbitrary languages to be parsed; it then applies rewrite rules to the abstract syntax trees that are generated. The commercial DMS software maintenance system [6] employs a similar method for its program transformations.

The Stratego language [36] also uses rewrite rules on abstract syntax trees; in Stratego, the rewriting strategy is user-definable. Stratego itself does not include facilities to parse and format the syntax trees; these are provided by auxiliary tools such as XT [37].

In addition, rewriting languages are often used by mathematics packages that are capable of symbolic calculus, such as Mathematica [76] and the GNU Emacs Calculator. The programming language Rigal [5], as well as logic programming languages such as Prolog, also employ rewriting.

```
%{
#include <stdio.h>
#include <ctype.h>
%}
%glr-parser
%%

expression: term
| { printf("(+ "); } expression ’+’ term2
| { printf("(- "); } expression ’-’ term2;

term2: { printf(" "); } term { printf(")"); };

term: factor
| { printf("(* "); } term ’*’ factor2
| { printf("(/ "); } term ’/’ factor2;

factor2: { printf(" "); } factor { printf(")"); };

factor: num | ’(’ expression ’)’;

num: digit | digit num;

digit: ’0’ { printf("0"); } | ’1’ { printf("1"); }
| ’2’{printf("2");} | ’3’{printf("3");} | ’4’{printf("4");}
| ’5’{printf("5");} | ’6’{printf("6");} | ’7’{printf("7");}
| ’8’{printf("8");} | ’9’{printf("9");};

%%

int yylex () {
  int c=’ ’; while(isspace(c)) c=getchar();
  if (c==EOF) return 0; else return c;
}
int yyerror (char* s) { fprintf(stderr,"%s\n",s); }
main() { yyparse(); }
```

Figure 3.8: A flawed attempt to correct Figure 3.7



```
Parse Tree (80 nodes)
  PhoneBook
    {PhoneBookLine}
      PhoneBookLine
        Name
          Lastname
            alpha = "Adams"
          "," = ","
          Firstname
            alpha = "John"
          MiddleInit
            alpha = "Q"
          "." = "."
```

```
Adams, John Q.  (900) 555-1234
Smith, John Q.  (800) 765-4321
Doe, John       (999) 999-9999
Doe, Jane J.    (000) 012-3456
```

Tree nodes, sample data, and grammar symbols are all cross-referenced

```
PhoneBookLine ::= Name PhoneNumber;

// Each line in the file consists of a name and phone number.

Name ::=        Lastname
                ","
                Firstname
                [MiddleInit "."]  // this term is optional
                ;
// A 'Name' consists of a Lastname, a literal comma, a Firstname, a
```

Figure 3.9: Testing a grammar in the ProGrammar IDE

```
define program
    [expression]
end define

define expression
    [expression] + [term] | [expression] - [term]
    | [term] | [output_expression]
end define

define output_expression
   ( + [expression] [expression] ) | ( - [expression] [expression] )
 | ( * [expression] [expression] ) | ( / [expression] [expression] )
end define

define term
    [term] * [factor] | [term] / [factor] | [factor]
end define

define factor
    [number] | ( [expression] )
end define

rule main
    replace [expression] OldE [expression]
    construct NewE [expression]
       OldE [convert_add] [convert_subtract]
            [convert_mult] [convert_div] [convert_factor]
    where not NewE [= OldE]
    by NewE
end rule

rule convert_add
    replace [expression]
        A [expression] + B [term]
    by ( + A B )
end rule

rule convert_subtract
    replace [expression]
        A [expression] - B [term]
    by ( - A B )
end rule

rule convert_mult
    replace [expression]
        A [term] * B [factor]
    by ( * A B )
end rule

rule convert_div
    replace [expression]
        A [term] / B [factor]
    by ( / A B )
end rule

rule convert_factor
    replace [expression]
        ( E [output_expression] )
    by E
end rule
```

Figure 3.10: Infix to postfix in TXL

Suitability. Rewriting works best for mathematical structures where the rewriting rules follow naturally from the mathematical definition of the structure. Any transformation that can be expressed informally as a set of "this pattern should be re-written as that" statements, which capture the complete transformation and are not merely examples of it, is likely to be easily implemented in a rewriting system so long as the input data can be parsed into the system. Rewriting systems may be more difficult to use in cases where it is less obvious what the rules should be, or when the transformation needs to go through one or more intermediate states before the desired result can be achieved—this needs more thought on the part of the transformation programmer.

3.5 XML-based transformation systems

There is no shortage of books and websites on XML [78] and related tools for presenting it, many of which make use of the XSLT transformation language [77]. This is effectively a variation on the rewriting systems mentioned above, except that it is not necessary to construct a specialised parser for each type of input before the rewriting can begin, because well-written XML makes the relevant structure explicit. Cascading Style Sheets (CSS) provides a more limited way of transforming XML text for presentation (CSS level 2 and above can be used with arbitrary XML).

The main disadvantage of XML-based transformation code is that it can be verbose—common transformation tasks can take a lot of code to express. This is also true of some non-XML frameworks. It can present problems, particularly for people with print disabilities, due to the overhead of writing and navigating the code. This can be partly alleviated by XML-aware editors and other development tools, so long as these applications themselves are accessible.

3.6 Matrix-like structures and multiple hierarchies

As shown in this chapter, most symbolic data is hierarchical, or *tree-like*, at least after it has been parsed. Generalised markup languages, such as XML [78], can be used for describing hierarchical structures over documents and data directly. In these, a piece of data can be enclosed within an "element", which can in turn be a member of a higher-level element, and so on.

It is often overlooked that much data is also *matrix-like* in nature, that is, it can be indexed along two or more orthogonal dimensions which can be addressed independently. Tables and spreadsheets are matrices. A musical score, which represents parallel streams of events, is matrix-like, and so are parallel translations of literary works—they can be interpreted as having tree-like structures, but there are several equally-valid branching orders. Often it is possible to read a document in several different ways, using, in effect, several different methods of indexing into the items of data that make up the document. It can be useful to treat these indices as the different dimensions of a multi-dimensional matrix, so that switching from one system to another amounts to slicing along a different dimension.

However, not everything makes sense as a matrix. Sometimes it is better to use a conventional tree-like hierarchical structure, particularly if the data's structure is recursive

```
<SCORE>                                 <SCORE>
  <PART>                                  <BAR>
    <BAR> a </BAR>                          <PART> a </PART>
    <BAR> b </BAR>                          <PART> c </PART>
  </PART>                 ⟷               </BAR>
  <PART>                                  <BAR>
    <BAR> c </BAR>                          <PART> b </PART>
    <BAR> d </BAR>                          <PART> d </PART>
  </PART>                                 </BAR>
</SCORE>                                </SCORE>
```

If a musical score is to be represented hierarchically then it can be structured either with many bars (measures) within each part or with many parts within each bar. It is sometimes necessary to transform between these two representations as part of a larger transformation. While this transformation can be accomplished using term rewriting or systems such as XSLT, the resulting code is long and complex.

Figure 3.11: An example of an enforced hierarchical structure that can impose artificial restrictions on working with matrix-like data

(as is the case with mathematical expressions, which can contain other expressions to any depth). But using a hierarchical structure throughout makes things more complex when they would perhaps be better represented as matrices—when a single hierarchical structure is not the most natural way to represent the structure of a notation, it can impose artificial restrictions on the notation's transformation, as shown in Figure 3.11.

Another challenge is when there are multiple, independent hierarchies over the same data; examples where data can be divided in more than one manner include:

- Any representation of a typeset document that includes layout data; the document can be divided into physical units (such as pages and columns) as well as into logical units (such as paragraphs), and it may be necessary to refer to both when making transcriptions in alternative notations—for example, Braille transcriptions of printed books are often required to refer to the original printed layout, for use when collaborating with a print user.

- Diagrams and other graphical data can be marshalled (serialised) in many different ways.

- Poetry and religious writing can have line numbers or chapters and verses, which are often independent of paragraphs and sentences.

- Any educational material that is printed in a diverse range of formats should have an indexing system that is independent of the physical formats, for use in a diverse classroom.

- In linguistics and natural language processing, there is often more than one possible parse over the same sentence, and these overlap each other.

Some programmers have represented multiple overlapping structures in a single hierarchy by making use of such things as XML linking, but again this can be complex and requires more effort from the programmer. The handling of matrix-like and overlapping structures can be viewed as a challenge for many existing transformation frameworks.

3.7 Summary

This chapter discussed programmable transformation systems, which can be used for implementing new transformation tasks as needed. It included discussion of Unix tools, parser generators, re-writing systems, and XML. It finished with a discussion of matrix-like and overlapping data structures as a limitation of many frameworks.

4 The 4DML transformation system

4DML (four-dimensional markup language) is my generalised transformation framework. This chapter introduces the framework and describes its design. Examples of its use are shown in the next chapter.

Chapter overview. The chapter begins by giving an overview of the 4DML transformation framework. Each component is then discussed in turn: the 4DML data structure, the transformation process, and the auxiliary languages CML (Compact Model Language) and MML (Matrix Markup Language). The chapter then elaborates further on the transformation process by discussing the use of various parameters in the 4DML model. This is followed by a discussion of the 4DML system's facilities for error and consistency checking, and alternative ways of using and understanding 4DML.

4.1 Overview of the 4DML framework

The 4DML framework consists of four main components:

1. The 4DML language itself, which is a generalised markup language for representing structured data, and can be compared with other generalised markup languages such as XML (Section 3.5),

2. *Matrix markup language* (MML), which is another generalised markup language designed to facilitate the input of multi-dimensional data,

3. A transformation tool that takes XML or MML as input, uses 4DML as an internal representation, and produces output in any text-based language by following a *model* of the desired structure,

4. *Compact model language* (CML) for representing the model (models may also be represented using XML).

As shown in Figure 4.1, data in XML or MML is first converted into 4DML—a process which needs no external information as XML and MML are both self-describing formats—and then transformed into any text-based output language under the direction of a model. The entire process may be surrounded by other transformations, such as a minimal amount of preprocessing (rarely needed) and the passing of the output through a typesetting system (more common), which are not part of the framework itself.

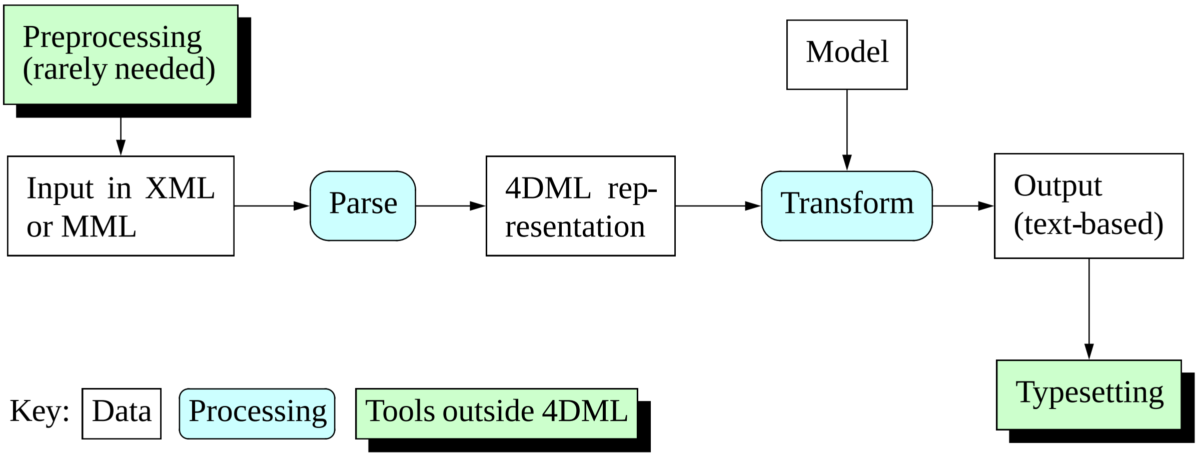


Figure 4.1: Overview of the 4DML transformation framework

4.2 4DML's representation of structured data

The 4DML space contains symbolic data, and a hierarchical structure over that data that indicates how the original input was parsed. Non-hierarchical structures are also possible and are described later.

Figure 4.2 shows a possible parse tree for a mathematical equation, represented by XML in Figure 4.3. Figure 4.4 illustrates how 4DML represents this parse tree. In Figure 4.4 (a), the symbols of the equation are written linearly on the bottom row, and each piece of markup is illustrated above the symbols and other markup that it encloses. Thus the entire set of symbols is part of an "equation" at the top of the parse tree; this equation is formed of two "expressions" and a "relation", and these contain different sets of the equation's symbols; the first "expression" is formed of three "terms" and two "operators", and so on.

Four-dimensional. Figure 4.4 (b) sub-divides the higher levels of the parse tree into repeated units. Each of these can now be described by a 4-tuple with components corresponding to

- the type of markup, i.e. the text in the coloured box,
- the position of each piece of markup among its peers, as illustrated by the background colour and numerical subscript of each box,
- the depth of the markup, corresponding to the vertical positioning of the box,
- the symbol that is being marked up, corresponding to the horizontal axis of Figures 4.4 (a) and (b)—the value is uniquely identified where necessary so that there is enough information to reconstruct Figure 4.4 (b).

Left-to-right order irrelevant. The left-to-right order of the symbols in Figure 4.4 (b) is not important; it is shown here to correspond to the original equation merely for

$$x^2 + 3x + 4 = 0$$

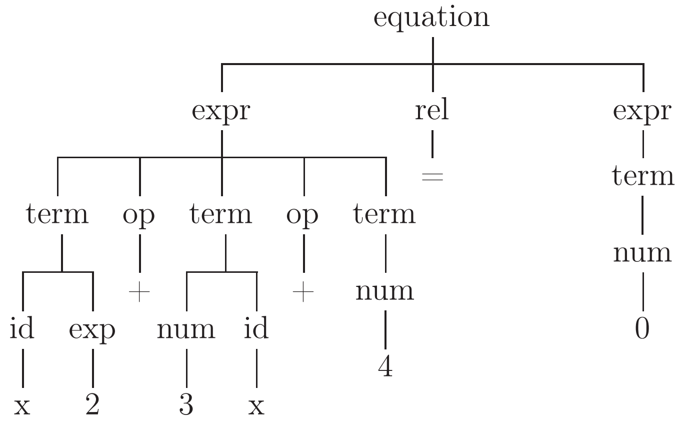
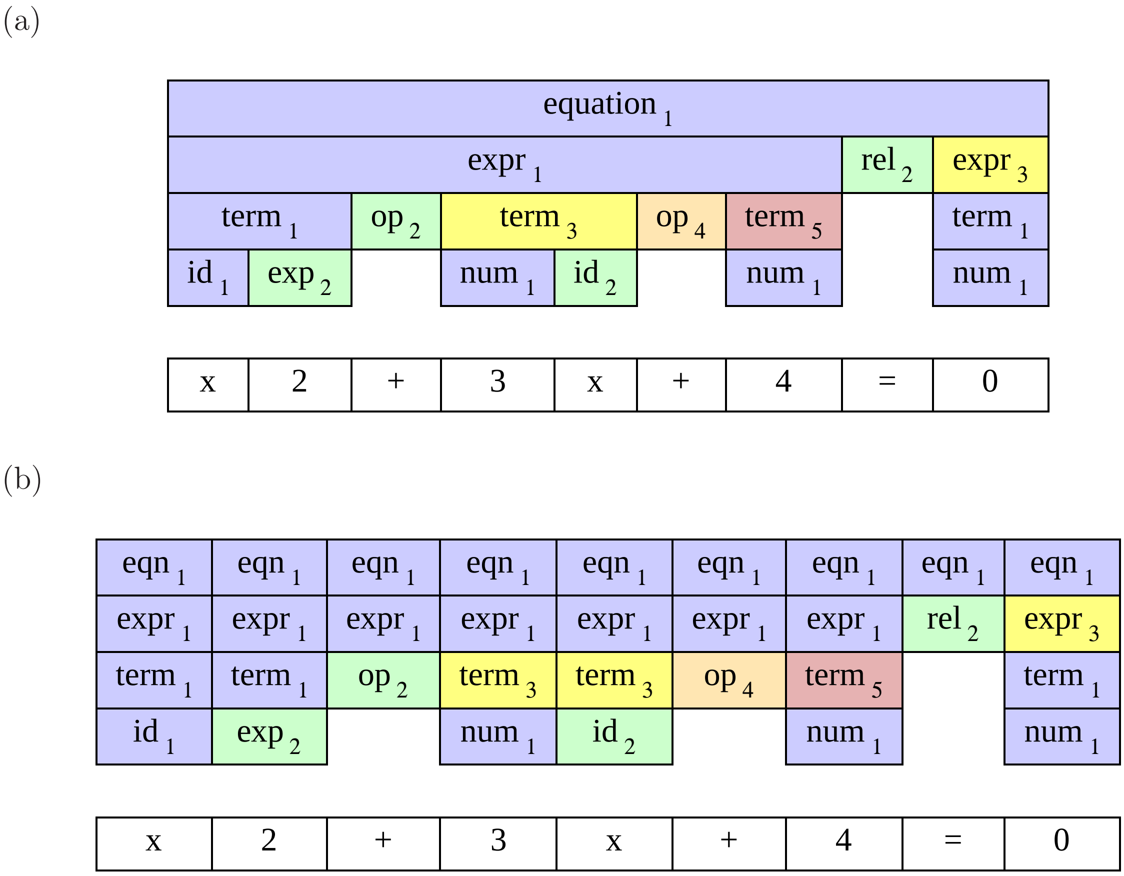


Figure 4.2: A mathematical equation, and a possible parse tree for it

```
<?xml version="1.0" ?>
<equation>
  <expr>
    <term>
      <id> x </id>
      <exp> 2 </exp>
    </term>
    <op> + </op>
    <term>
      <num> 3 </num>
      <id> x </id>
    </term>
    <op> + </op>
    <term>
      <num> 4 </num>
    </term>
  </expr>
  <rel> = </rel>
  <expr>
    <term>
      <num> 0 </num>
    </term>
  </expr>
</equation>
```

Figure 4.3: XML's representation of the parse tree in Figure 4.2


(a)
equation$_1$
expr$_1$ rel$_2$ expr$_3$
term$_1$ op$_2$ term$_3$ op$_4$ term$_5$ term$_1$
id$_1$ exp$_2$ num$_1$ id$_2$ num$_1$ num$_1$
x 2 + 3 x + 4 = 0
(b)
eqn$_1$ eqn$_1$ eqn$_1$ eqn$_1$ eqn$_1$ eqn$_1$ eqn$_1$ eqn$_1$ eqn$_1$
expr$_1$ expr$_1$ expr$_1$ expr$_1$ expr$_1$ expr$_1$ expr$_1$ rel$_2$ expr$_3$
term$_1$ term$_1$ op$_2$ term$_3$ term$_3$ op$_4$ term$_5$ term$_1$
id$_1$ exp$_2$ num$_1$ id$_2$ num$_1$ num$_1$
x 2 + 3 x + 4 = 0

(c)
(num, 1, 4, "0"), (expr, 1, 2, +^{2}), (expr, 1, 2, +^{1}), (term, 3, 3, x^{2}), (num, 1, 4, "4"), (term, 5, 3, "4"), (equation, 1, 1, +^{2}), (equation, 1, 1, +^{1}), (op, 2, 3, +^{1}), (id, 1, 4, x^{1}), (id, 2, 4, x^{2}), (equation, 1, 1, =), (equation, 1, 1, "3"), (term, 1, 3, "0"), (term, 1, 3, "2"), (expr, 1, 2, x^{2}), (expr, 1, 2, "3"), (term, 1, 3, x^{1}), (term, 3, 3, "3"), (op, 4, 3, +^{2}), (num, 1, 4, "3"), (exp, 2, 4, "2"), (expr, 1, 2, "2"), (equation, 1, 1, x^{1}), (equation, 1, 1, x^{2}), (expr, 3, 2, "0"), (expr, 1, 2, "4"), (equation, 1, 1, "4"), (equation, 1, 1, "2"), (equation, 1, 1, "0"), (expr, 1, 2, x^{1}), (rel, 2, 2, =)

Figure 4.4: 4DML's representation of the parse tree in Figure 4.2

illustrative clarity. Should any aspect of the original left-to-right order be needed, it can be retrieved from the position and depth information; otherwise, 4DML is free to sort the symbols into a different order as required by the output notation.

The absence of a "global" left-to-right ordering also helps with representing multiple independent sets of markup over the same data. In this case, the order in which the data is to be read might depend on *which* set of markup is in use. So the ordering should be a property of the markup, not of the underlying data—the markup is effectively a system for indexing into the data.

4.3 Transformation by model

The primary algorithm associated with 4DML is "transformation by model", which takes some 4DML input along with a "model" of the desired structure, and transforms the input as necessary to reflect the structure of the model. The algorithm can also report which objects have been lost in the process, if any.

The algorithm works by performing a top-down traversal of the model, and reads off relevant parts of the input as it does so. Since 4DML can be read in many different ways, it is not difficult to read it in whatever way is dictated by the structure of the model, regardless of whether or not this matches the original structure. Hence it is possible to perform complex structural transformations merely by writing down the form of the desired result; it is also possible to follow the structure of the input like a conventional stylesheet processor.

In other words, each element in the model will cause the following things to occur:

1. The input is searched for all elements that match the name of the model element. Only such elements at the highest level at which they occur will be used. The search will cut across all other markup.

2. The input is divided into groups, one for each distinct element that was found, and the groups are sorted by position number. Any other markup from the input will be included in each group.

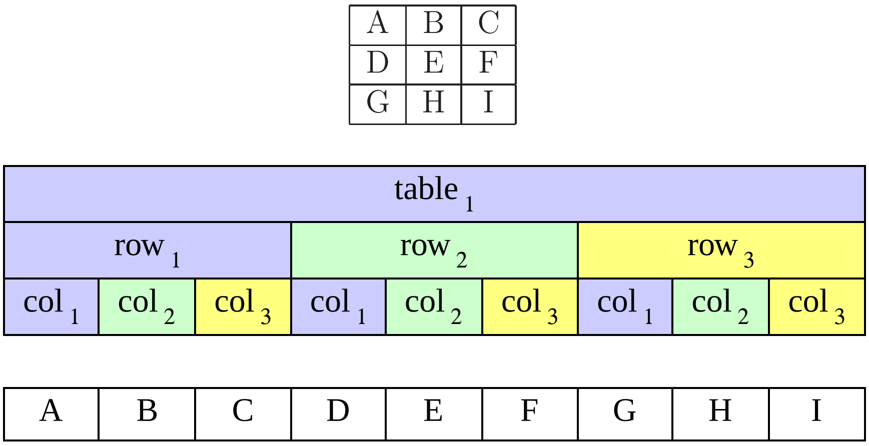
3. Any model code that occurs within the current model element will be executed once for each group.

4. If the model element is empty (a leaf node), then the data from each group is copied to the output, discarding all remaining markup.

Thus an element X in the model is effectively saying "for each X". Arbitrary text in the model is copied to the output whenever it is encountered. Model elements can be given attributes (parameters) to specify their behaviour in a more flexible manner; this is further described later in the chapter.

4.3.1 Examples

Table transposition Figure 4.5 shows how a 3×3 table or matrix might be represented in 4DML, if the input format represents columns within rows (as is the case with HTML).



A	B	C
D	E	F
G	H	I

Figure 4.5: A 3×3 table and an illustration of how it might be represented in 4DML

It is equally easy to select a row as to select a column (Figures 4.6 and 4.7), so if the user wishes to transform the table into a notation that requires it to be read in columns rather than rows then this is virtually transparent.

The complete specification of the order in which the transposed table is to be written is essentially the nested statement:

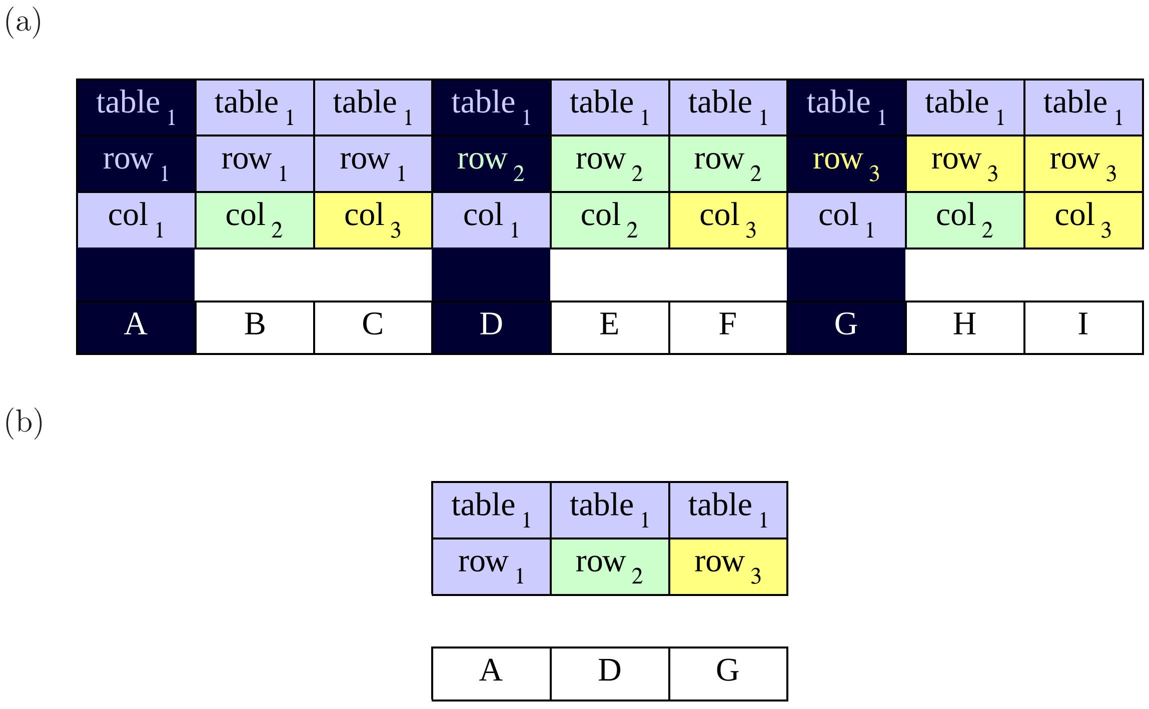
```
for each "col" $c$, select $c$ and
  for each "row" $r$, select $r$ and
    Write out what is selected
```

In 4DML's compact model language (CML), this is abbreviated to `col/row` (more correctly, `table/col/row`, which would properly handle the case of several tables), and its result is shown in Figure 4.8—the symbols are now in the correct (transposed) order, but the markup needs adjusting. This is achieved by using the slightly-modified CML model `table/col rename=row/row rename=col` (or, equivalently, `table/row from=col/col from=row`) (Figure 4.9).

More complex example. Figure 4.10 shows how 4DML might represent "distributed music encoding", which will be discussed in Section 5.5.1 on page 92, and Figure 4.11 shows how this might need to be re-arranged for a music typesetting program—this was generated from the CML model `score/bar/part/note/(pitch,length)`. A more complete model and its output are shown in the next section.

4.4 Compact Model Language (CML)

CML is a text-based language designed to facilitate the brief coding of 4DML models. In practice, most models have a tail-recursive structure; they express such things as "for each bar, for each part, for each note, ..." which in XML would require a number of closing tags:



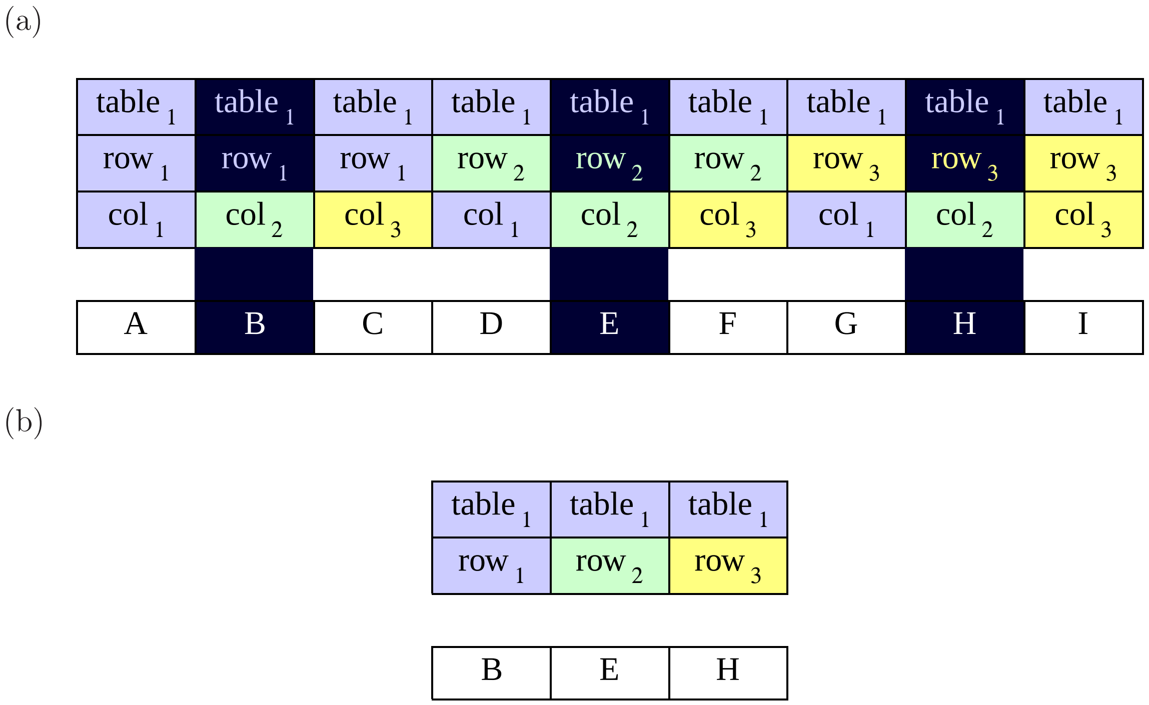
(a)

table$_1$	table$_1$	table$_1$	table$_1$	table$_1$	table$_1$	table$_1$	table$_1$	table$_1$
row$_1$	row$_1$	row$_1$	row$_2$	row$_2$	row$_2$	row$_3$	row$_3$	row$_3$
col$_1$	col$_2$	col$_3$	col$_1$	col$_2$	col$_3$	col$_1$	col$_2$	col$_3$
A	B	C	D	E	F	G	H	I

(b)

table$_1$	table$_1$	table$_1$
row$_1$	row$_2$	row$_3$
A	D	G

Figure 4.6: (a) selecting column 1, (b) the resulting subset



(a)

table$_1$	table$_1$	table$_1$	table$_1$	table$_1$	table$_1$	table$_1$	table$_1$	table$_1$
row$_1$	row$_1$	row$_1$	row$_2$	row$_2$	row$_2$	row$_3$	row$_3$	row$_3$
col$_1$	col$_2$	col$_3$	col$_1$	col$_2$	col$_3$	col$_1$	col$_2$	col$_3$
A	B	C	D	E	F	G	H	I

(b)

table$_1$	table$_1$	table$_1$
row$_1$	row$_2$	row$_3$
B	E	H

Figure 4.7: (a) selecting column 2, (b) the resulting subset

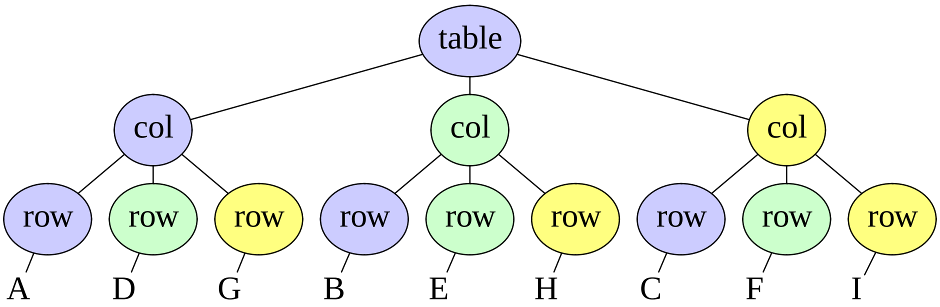


Figure 4.8: Table transposition without markup adjustment

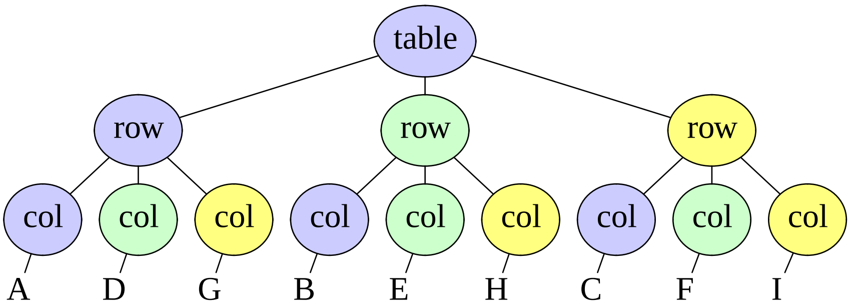
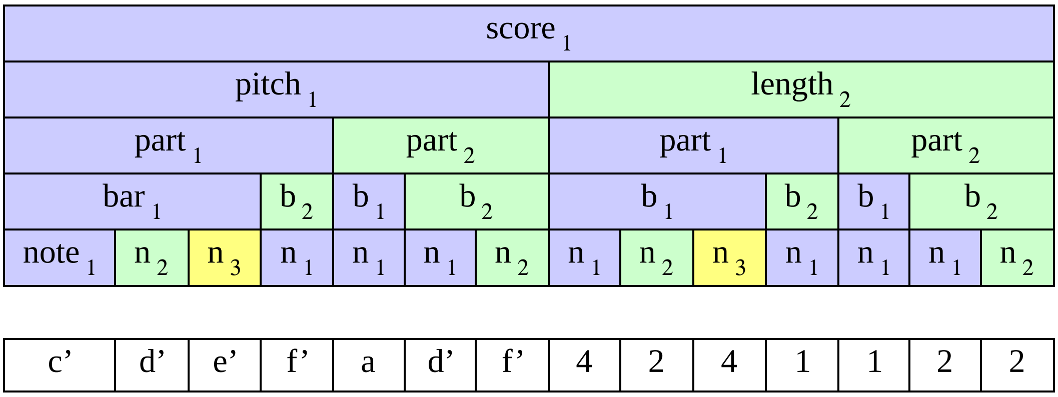


Figure 4.9: Table transposition with markup adjustment



score $_1$													
pitch $_1$							length $_2$						
part $_1$				part $_2$			part $_1$				part $_2$		
bar $_1$			b $_2$	b $_1$	b $_2$		b $_1$			b $_2$	b $_1$	b $_2$	
note $_1$	n $_2$	n $_3$	n $_1$	n $_1$	n $_1$	n $_2$	n $_1$	n $_2$	n $_3$	n $_1$	n $_1$	n $_1$	n $_2$

c’	d’	e’	f’	a	d’	f’	4	2	4	1	1	2	2

Figure 4.10: Possible 4DML representation of distributed music encoding

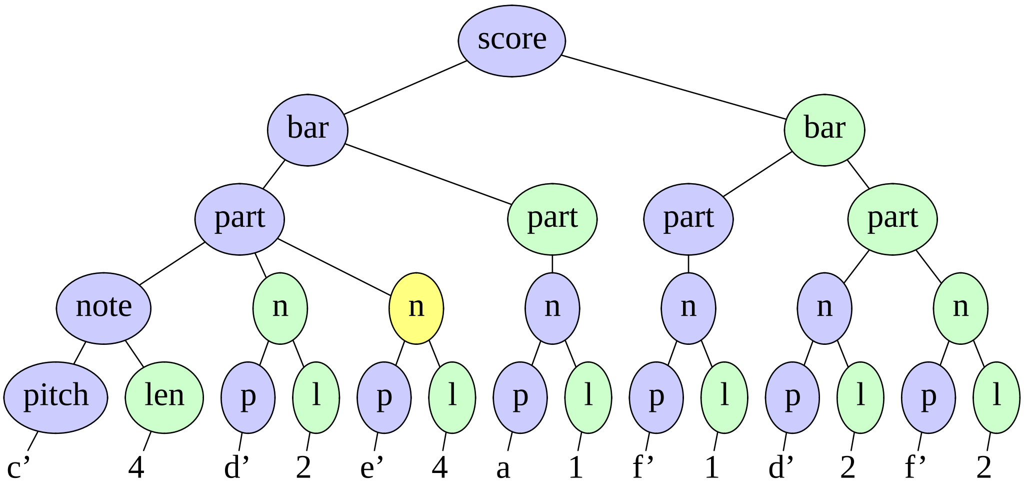


Figure 4.11: A possible re-arrangement of Figure 4.10

```
<bar> <part> <note> </note> </part> </bar>
```

In CML, the above is expressed as `bar/part/note`. CML also has other operators and can represent any hierarchical document, but its syntax (Figure 4.12) is particularly designed for representing typical 4DML models concisely.

Complete models in CML are often small enough to be given to the processor as command-line arguments. Alternatively, CML can be embedded into a file as a macro language (a little like PHP), which is useful when there is a large amount of text to include before, after, or between the input, as in the example in Figure 4.13, the result of which is shown in Figure 4.14.

As with the language BRL [47], CML's embedding syntax is identical to its "arbitrary text" delimiters; hence it is possible to switch out of the CML anywhere where arbitrary text is to be inserted. The delimiters can be changed if necessary.

4.5 Matrix markup language (MML)

Before 4DML can work with structured data, the data must first be made available to it. It can be cumbersome to hand-code matrix-like data in a hierarchical markup language like XML, since the markup is very verbose and repetitive. For example, in coding music, one might have to enclose each note in a `<NOTE>`...`</NOTE>` pair, whereas it would be easier to define a separator (for example, '`-`') to stand for "next note" (other separators can advance the bar or part). In the general case, one can construct a parser for an arbitrary domain-specific input language and convert it into XML, but this can be a significant amount of effort for an end-user, and there is scope for a markup language that provides for some simple re-definitions (such as "`-` means next note") while not being as complex as a parser generator.

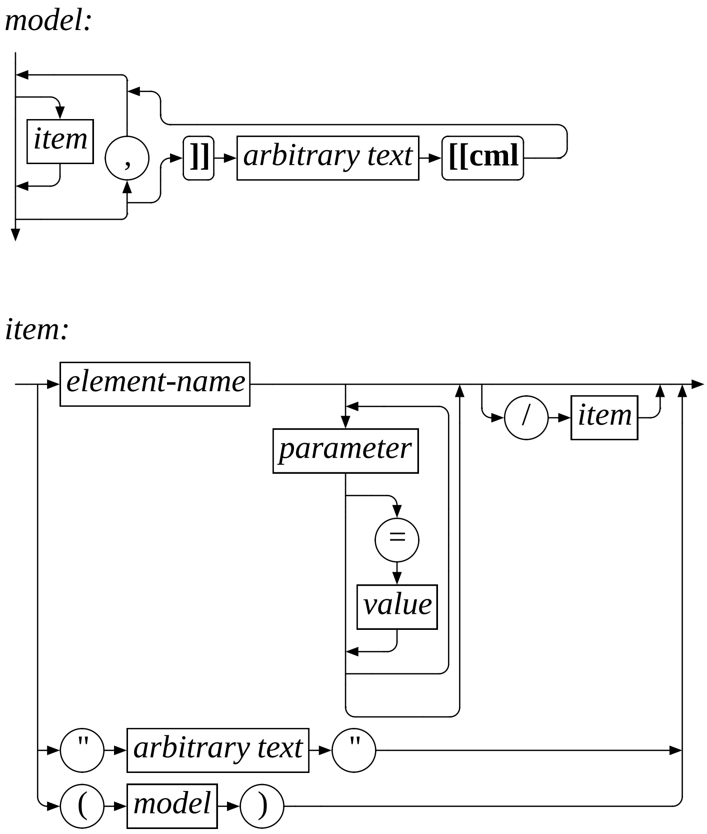


Figure 4.12: Syntax of CML

```
\documentclass{article} \begin{document}
\begin[20pt]{lilypond} \key a \minor
  [[cml bar between="|" /
   part before="<<" after=">>" between="\\" /
    ("{", note between=" "/(pitch,length), "}")
  ]]
\end{lilypond} \end{document}
```

Figure 4.13: Embedding CML into another language

```
\documentclass{article} \begin{document}
\begin[20pt]{lilypond} \key a \minor
  <<{c'4 d'2 e'4}\\{a1}>>|<<{f'1}\\{d'2 f'2}>>
\end{lilypond} \end{document}
```

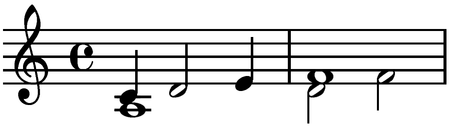


Figure 4.14: Output from Figure 4.13 and its typeset result

Matrix Markup Language (MML) is a text-based language that can represent structure in several ways, such as by using `begin`/`end` pairs, `element: value` lines, and matrix-like blocks such as the one in Figure 4.16, the result of which appears in Figure 4.10. MML's syntax is shown in Figure 4.15.

Each matrix-like block has two parts, a header and a body. The first blank line separates the header from the body. The header specifies the meaning of the separation symbols that are to be used in the body. The following constructs can appear in the header:

1. An optional name for an element that will enclose the whole block,

2. `Have`...`as` constructs. `Have` *A* `as` *B* causes the string *A* to be used as a separator of elements named *B*. If there are several `have`...`as` constructs then the resulting elements are nested in the order of the constructs, so the *last* construct has the separator of *highest* precedence and the name of the *innermost* elements. If desired, multiple constructs can be amalgamated thus: `have` A_1 A_2 A_3... `as` B_1 B_2 B_3...

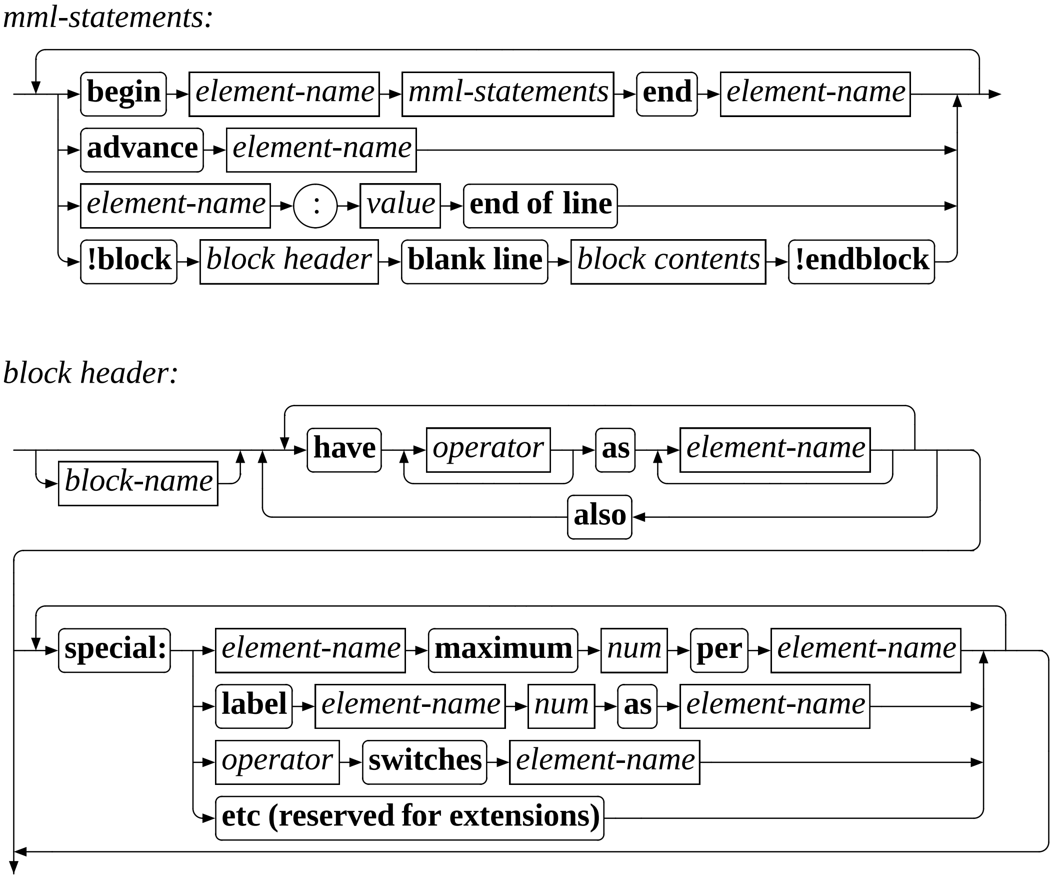
   As a special case, the separator `paragraph` means one or more blank lines, `newline` means an end of line, `whitespace` means any other whitespace and `character` means the empty string (it results in the input being split into individual characters).

3. `also`—this can be used to separate independent groups of `have`...`as` constructs so that the block's body can represent multiple, independent hierarchies over the same data.

4. `special:`—can be used to specify special ways of interpreting some operators (rarely needed).

4.6 Other aspects of 4DML models

It is possible to extend the 4DML model language almost indefinitely by supporting more parameters that modify the processing behaviour. In keeping with 4DML's design principles, this should be kept to a minimum, and any such extensions should be orthogonal,



Notes:

1. `advance e` will advance the position of element `e` even if it is *not* the innermost element that is currently in progress
2. The syntax of block headers is explained in the text

Figure 4.15: Syntax of MML

```
begin score

!block pitch
have newline whitespace - as part bar note

c’-d’-e’ f’
a d’-f’
!endblock

!block length
have newline whitespace character as part bar note

424 1
1 22
!endblock

end score
```

Figure 4.16: Two matrix-like blocks in MML. The syntax is explained in the text.

not required learning. “Special case” extensions for handling particular transformation problems are avoided—4DML would be a poor generalised transformation framework if it required a special-case extension for each new problem.

The features that *were* implemented are described here.

4.6.1 The external stack

This is a mechanism that allows the model to access data outside the currently-selected subset. Consider a document with a “title” and one or more “parts” (Figure 4.17). A musical score is such a document. If it is desired to write out the parts individually, each one with its own copy of the title, then there is a problem—selecting any “part” will not include the title in the selection (Figure 4.18) and hence the title would not be available.

The external stack is a display stack of all data that is not processed. In this case, the operation “for each part” will process the data shown in Figure 4.19, so the rest of it (Figure 4.20) is added to the display stack for the duration of the “for each part” operation. When “title” is called for and none is found in the currently-selected subset, the stack is searched in reverse and the title is found there (Figure 4.21).

Occasionally it is necessary for the user to override the behaviour, for example, by specifying `external=never` for an operation so that the external stack is not searched, or `external=always` so that it is always searched (instead of the current subset). This is rare—the default behaviour (to search the external stack if nothing is found in the current subset) is usually correct.

Another possible override is `include-rest`, which causes the external stack entry (the “rest” of the data) to be included in each subset generated by the “for each” operation.

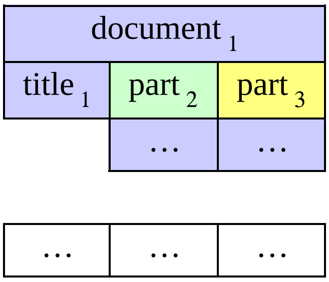


Figure 4.17: A document with parts and a title
(For an explanation of the numerical suffices, see page 47)

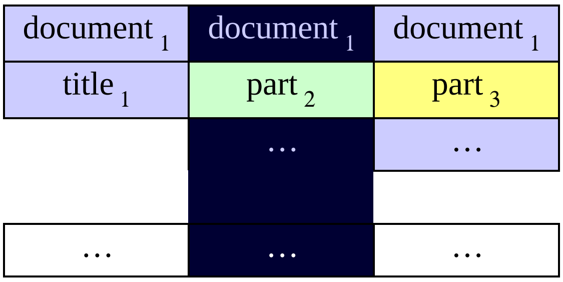


Figure 4.18: Selecting a “part” will discard the “title”

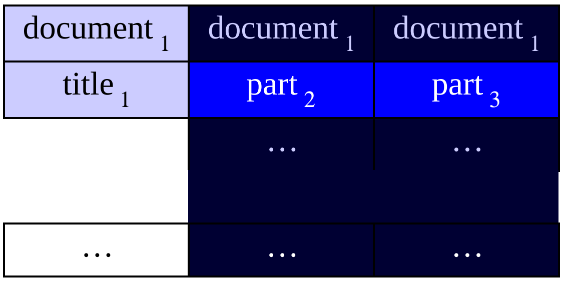
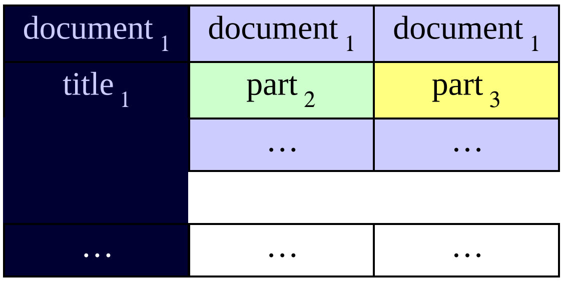
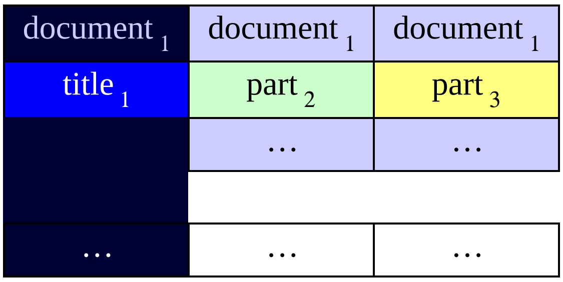


Figure 4.19: Data processed by “for each part”



document $_1$	document $_1$	document $_1$
title $_1$	part $_2$	part $_3$
	…	…
…	…	…

Figure 4.20: Data added to external stack by "for each part"



document $_1$	document $_1$	document $_1$
title $_1$	part $_2$	part $_3$
	…	…
…	…	…

Figure 4.21: Finding the title on the external stack

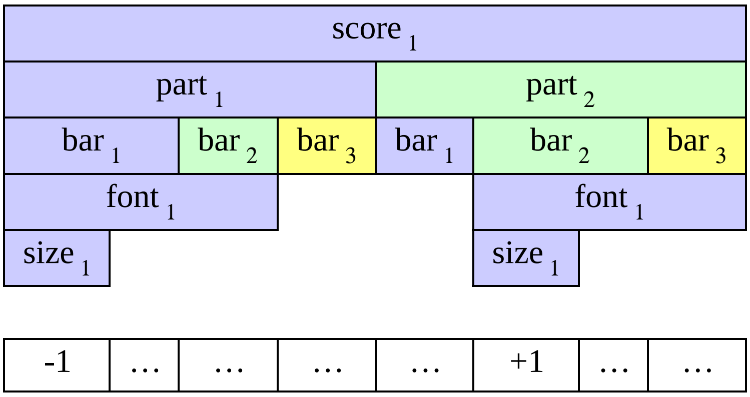


Figure 4.22: Musical data with added font markup

This is useful, for example, in a song with music and several verses, where the music is to be copied out with each verse—by saying `verse include-rest`, the model may then access, for each verse in turn, the lyrics of that verse in synchronisation with the music.

4.6.2 The "broaden" operation

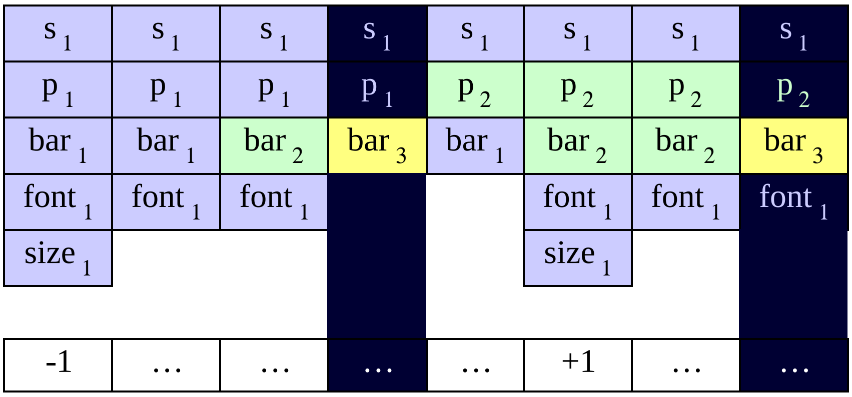
This is another mechanism for accessing data outside the current subset, and it has an application in link traversal.

Reading attributes

A simpler application for "broaden" is in the reading of attributes that are stored within the scope of some element that is only partially in the current subset. Consider for example Figure 4.22, which represents musical data with added font information (including attributes), which in this case is chosen at random for illustrative purposes. Interleaving the bars from both parts is trivial as has already been shown.

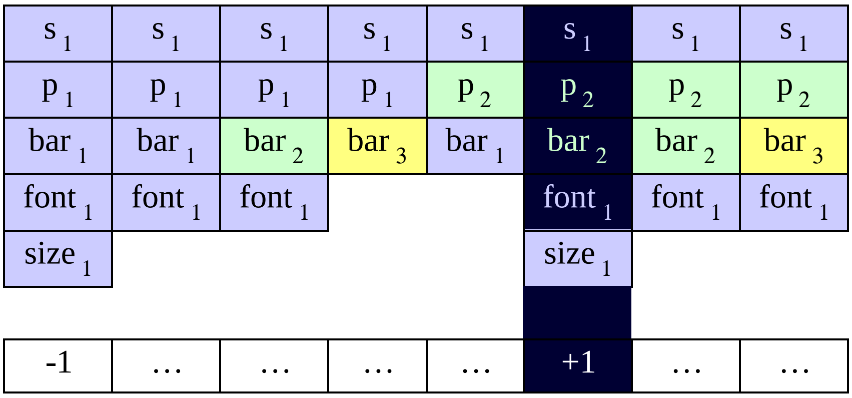
However, if the font markup is to be preserved, a problem arises—selecting the third bar (Figure 4.23) will discard the font information highlighted in Figure 4.24, so selecting the third bar and part 2 will not leave this information available (we will still have the "font" markup, but not its attributes). Even if the attributes were placed outside the "bar" markup so that they went into the external stack, it would still be difficult to select them unambiguously.

Specifying `font broaden` will select the entirety of the "font" element, not just what is visible in the current subset (Figure 4.25). This selects too much, due to 4DML's automatic combining of elements, so in this case it is necessary to broaden within a context—`font broaden=part` will ensure that the scope is only broadened within that of the current `part` element. Another solution is to ensure that the numbering of `font` does not begin again at the change of `part`, but this might not be possible if the data has been merged from separate sources.



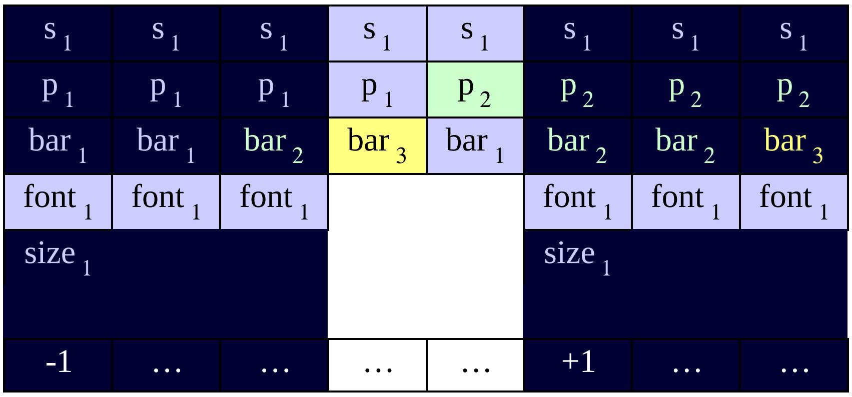
s_1	s_1	s_1	s_1	s_1	s_1	s_1	s_1
p_1	p_1	p_1	p_1	p_2	p_2	p_2	p_2
bar $_1$	bar $_1$	bar $_2$	bar $_3$	bar $_1$	bar $_2$	bar $_2$	bar $_3$
font $_1$	font $_1$	font $_1$			font $_1$	font $_1$	font $_1$
size $_1$					size $_1$		
-1	…	…	…	…	+1	…	…

Figure 4.23: Selecting the third bar



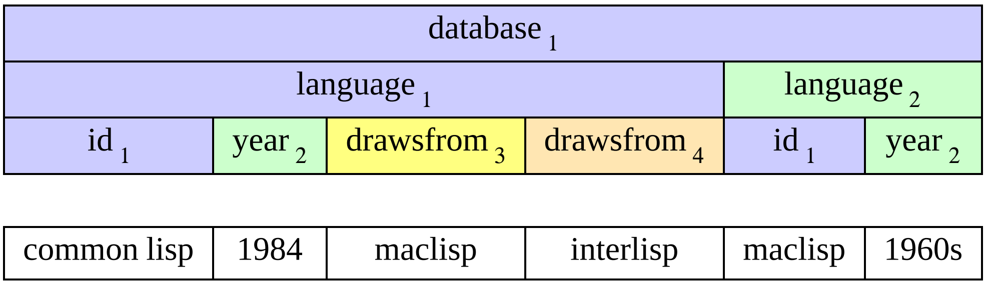
s_1	s_1	s_1	s_1	s_1	s_1	s_1	s_1
p_1	p_1	p_1	p_1	p_2	p_2	p_2	p_2
bar $_1$	bar $_1$	bar $_2$	bar $_3$	bar $_1$	bar $_2$	bar $_2$	bar $_3$
font $_1$	font $_1$	font $_1$			font $_1$	font $_1$	font $_1$
size $_1$					size $_1$		
-1	…	…	…	…	+1	…	…

Figure 4.24: Attribute that is discarded in Figure 4.23



s_1	s_1	s_1	s_1	s_1	s_1	s_1	s_1
p_1	p_1	p_1	p_1	p_2	p_2	p_2	p_2
bar $_1$	bar $_1$	bar $_2$	bar $_3$	bar $_1$	bar $_2$	bar $_2$	bar $_3$
font $_1$	font $_1$	font $_1$			font $_1$	font $_1$	font $_1$
size $_1$					size $_1$		
-1	…	…	…	…	+1	…	…

Figure 4.25: Data selected by `font broaden`



database $_1$					
language $_1$				language $_2$	
id $_1$	year $_2$	drawsfrom $_3$	drawsfrom $_4$	id $_1$	year $_2$
common lisp	1984	maclisp	interlisp	maclisp	1960s

Figure 4.26: Data with an implicit link

Link traversal

Another application of "broaden" is in link traversal. Consider Figure 4.26, an extract from data on historical relationships between programming languages. There is an implicit link between the two instances of "maclisp". If it is desired to output a subset of the data that consists of related programming languages, then the links need to be traversed.

To represent the implicit link in 4DML structure, an element is added that covers both instances of "maclisp" (Figure 4.27). This was done automatically by linking *any* duplicate strings—since this markup is independent, it can be ignored if necessary. The link is then traversed with the model:

```
drawsfrom/link broaden/id/language broaden
```

This process is illustrated in Figure 4.28—first the link is selected and broadened, then something is selected to indicate which "end" of the link is required (in this case "id"), then the context of this is broadened as required. Note that all links can be treated as bi-directional or many-to-many mappings if desired—traversing links in reverse is no more

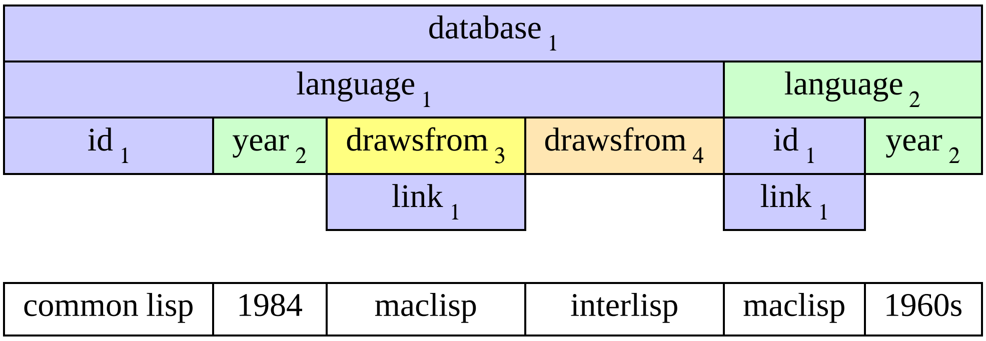


Figure 4.27: Representing a link in 4DML

difficult than forwards traversal, and more complex operations (such as jumping directly to other documents that cite the same material as the current one) are also conceivable.

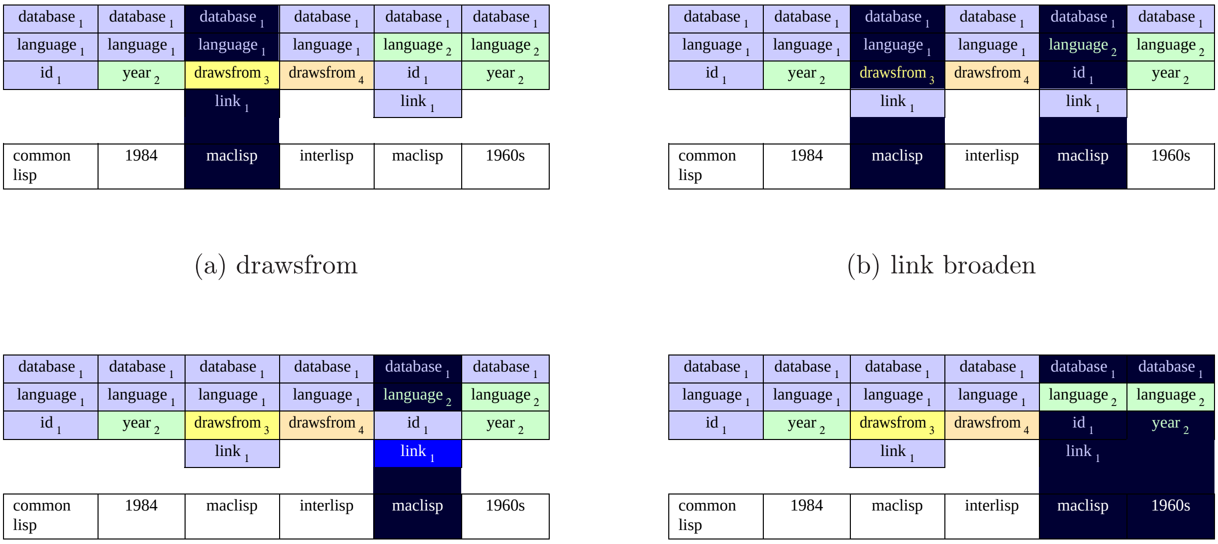
4.6.3 Naming the output structure

XML output. Elements can be renamed, either with the `rename` parameter, which specifies the new name, or with `from`, which specifies the old name and indicates that the element name itself is the new name. Elements can also be removed altogether, their contents being output without them. New elements that do not occur in the input can be added by the model (`no-input` specifies an element that should not be looked for in the input), and elements can be converted to attributes.

When the *input* is XML, elements with empty names are added around XML cdata, so that position information is preserved when cdata is mixed with other child elements. XML attributes are represented as children of a child element named `!attributes` (not valid in XML) which does not disturb the position numbers of the other children.

Arbitrary text. When the output language is not XML, it is useful to add arbitrary markup text as needed. This can be included unconditionally in the model—it is copied to the output whenever it is encountered—or text can be specified to be output at the `start` or `end` of an element (for as many times as that element occurs) or `before`, `after` or `between` all the occurences of the element.

Counting. One feature of XSLT which was useful to include in 4DML is the ability to add numbers to the output, so as to count explicitly such things as the sections in a document or the verses in a song. In 4DML, the number(s) supplied by `count` (which outputs a count of the given element) can refer either to the *original* element positions, or (if `renumber` is active) to the order in which they are processed, which might not match the original if a drastic re-ordering has taken place or if a subset has been taken (Figure 4.29). In most real examples, however, the numbers will be unchanged. (See Section 6.2.2 for further discussion of this design decision.)



(a) drawsfrom

(b) link broaden

(c) id

(d) language broaden

Figure 4.28: The effect of the model described in Section 4.6.2

Element name and position:	A_1	A_2	B_3	A_4	C_5	A_6	A_7	B_8	A_9
Output of `A count`:	1	2		4		6	7		9
Output of `A count renumber`:	1	2		3		4	5		6

Figure 4.29: The effect of `renumber`

If the desired numbering is related to a structure that is logically different from the one being processed, but the other structure is still present in the 4DML input, then the elements in the other structure can be referenced with `count if-changed`; informally, this means "see if our position in the other structure has changed and, if so, say where we are". This can be used, for example, to handle verse numbers or other indexing that is not strongly related to the actual structure. `if-changed` is actually independent from `count`; it can also assist the development of 4DML library routines which might need to inspect their context when called.

4.6.4 Order and conditions of processing

Sequential processing. Sometimes it is necessary to process input data in the order in which it arrives, like a more conventional transformation system, rather than grouping together elements of the same type. This is achieved by using the `sequential` parameter, which puts each element in its own subset and processes them separately. `sequential` can be used with `wildcard`, which causes *any* element to be matched; since `wildcard` implies `no-strip`, each matched element is left in the subset and can then be matched by one or more element-specific parts of the inner model (like a C "switch" on the element type).

`sequential` implies `children-only`, which ensures that, in line with standard rewriting methods, the directive does not include markup higher up the tree in the subsets that it generates (such markup is normally included so that transpose-like operations will work).

Recursion and subroutines. `call` is a means of adding recursion to the model. Each model element is also treated as a named subroutine (procedure), so specifying a model element (such as "expression" in a mathematics transformation) has the side-effect of defining a procedure of the same name ("expression"). The contents of this procedure will be that of the model element, and its name will be in scope (unless overridden) throughout the contents of the model element itself, so that recursion is possible, and also throughout subsequent model elements, so that non-recursive calling is possible, up to the end of the parent (enclosing) model element. Additionally, if the parent model element has the `export-code` parameter then the name will be in scope through one more level; this makes it possible to package small "libraries" of partial models.

Recursion will frequently be used when processing notations such as mathematics, where one "expression" can contain another (in an element `expression`, the CML code `expression call=expression` will recurse, i.e. treat this `expression` in the same way as the current or previously-mentioned `expression`). Non-recursive calling might be used when processing documents that have differently-named objects which are to be treated similarly; `a call=b` means "treat `a` as we treated `b`". This might be used in addition to some other treatment—`a/...`, `a call=b` will first perform the given processing on all `a`s, and then process them all again, this time treating them as `b`s. This can be combined with `sequential` (above) if each `a` is to have both treatments before the next is treated; however, it is often necessary to make a separate pass through the data, such as when making a table of contents.

Value-based and count-based restrictions. The parameters `value` and `other-values` can be used to construct a “switch” on an element’s *value* (the data contained by a leaf element); this would be used, for example, when a translation table is required. Similarly, `total` can be used if it is necessary to perform discretely different actions depending on how many instances of the given element are to be processed (this was needed to handle different cases of scaling in the koto typesetting in Figure 5.23).

The parameters `start-at` and `end-at` give numerical restrictions on the positions of the elements processed. This is often needed for “scraping” systems (Section 5.2). As with counting (above), this can refer either to the original positions or to the relative positions in the current subset depending on whether `renumber` is active. For convenience, `number` is equivalent to setting both `start-at` and `end-at`.

Reverse. A `reverse` parameter was added for completeness; it was only used when breaking the chords in the modern-style composition (Figure 5.18).

Merge. The `merge` parameter causes all elements to be merged and processed as one. This can be conceptually useful on occasions, such as in the weather example (Section 5.2).

4.7 Error checking

When data is provided in Matrix Markup Language (MML) it is possible that mistakes will be made in data entry. For example, one item of data might be missed, causing the rest of that row to be misaligned with the others. Although this is comparatively rare when the input language is tailored to the requirements of the individual user, it is still helpful if such errors are detected and reported by 4DML; if they are not, then they might cause the output language’s typesetter to crash without clear explanation, or they might result in incorrect output. Even if an error is detected during proof-reading, it can be difficult to locate its true source if it is not reported by 4DML.

4.7.1 “expected”

A comparatively simple way of detecting many errors is to make use of a model attribute `expected`, which indicates that the element named in the model here is *expected* to be in the currently-selected subset of the input. Normally, 4DML ignores model elements that are not found, since the model can specify what to do with certain elements “in the event that they exist”. However, if an element is “expected” then an error is reported if none can be found. (Alternatively, `expected` can be made the default and another attribute can be used to switch it off.)

Using `expected` in appropriate places can catch most errors like applying the wrong model to the wrong sort of data, such as trying to transform mathematics as though it were music, or omitting some vital part of the input. It can also catch many mismatches between row lengths and so forth—if the user has input two or more rows of data and the model reads each column (as is the case in distributed music encoding, which will be discussed in Section 5.5.1 on page 92), then if each column is ‘expected’ to have something from each of the row types that were input, there will be an error if one row is longer than the other, such as if the wrong number of separators were used in the MML file.

```
$ 4dml -minput songs.mml -cmodel basspart.cml > songbass.tex
4DML transformation utility v0.63 (c) 2003 Silas S. Brown.  No warranty.
Transforming...  error
Transform error:  Expected 'bass' but none found
Positions:  'document':  1, 'song':  4, 'chorus':  1, 'system':  1,
'word':  11
```

Figure 4.30: An error report from 4DML

The transformation tool can diagnose the likely location of the error by outputting counts (page 64) of all elements currently in progress, as shown in Figure 4.30. Further work could include improving this diagnostic to point out the likely location in the input file itself, although in practice the existing diagnostic is usually adequate.

Alternative use of `expected`. Models that use "switch"-like constructs (page 67) may use `expected` to assert that a particular case should never be matched. This can be done, for example, as follows:

```
    item value="X"/... ,
    item value="Y"/... ,
    item other-values / expert-use expected
```

The "`expert-use expected`" will look for an element `expert-use` and report an error if it is not found. This approach means that an "expert" can override the assertion by providing the `expert-use` element at that point in the input. It is useful for warnings that are not always errors; `expert-use` can be re-named to something more appropriate to each warning.

4.7.2 Confining the effects of errors

If the effect of an error is confined to the area in which the error occurred, then the error becomes easier to locate once detected. Such confinement is usually achieved by including more synchronisation information than is strictly necessary.

For example, if the user is copying from handwritten music manuscript, then it may be advisable to encode information about the page boundaries and perhaps the line breaks of the original manuscript. This information could be used to annotate the output with cross-references to the source material, but even if that is not desired, it provides confinement for errors—if each page is encoded separately then most errors will affect only the page that they are on. Following the form of the original also aids navigation around the unfinished encoding work. An additional benefit is that, when coding a sparse aspect of the music, some typing can be saved by skipping to the next synchronisation point (bar, line or page) when there is no more to be said before that.

Even if there is no original physical document, artificial divisions that provide synchronisation can often be included.

Aspect confinement. Although it may seem otherwise, aspect-oriented approaches of coding in MML can actually be advantageous when correcting errors, as the user need consider only one aspect when finding the error (if it is not detected automatically)—it is not necessary to navigate around large quantities of information about unrelated aspects that are not affected by the error. In music, for example, non-aspect-oriented music coding systems generally interleave many different kinds of information about the music, and it can take real effort to pick out the parts of the code that are relevant. This problem can be avoided in MML due to aspect separation.

Temporal effects. Another advantage of aspect separation in the case of musical notations is that, because the user is coding only one aspect of the music at a time, the amount of time required to type in each note is very short. This means that musicians can imagine the music playing as they type it, which is not normally possible if much time must be spent on the details of one note before moving on to the next one. If the music is "playing" in the musician's mind then this virtually eliminates synchronisation errors, since the process of data entry becomes more like that of playing an instrument or conducting. (For the same reason, it also becomes less tedious.)

4.7.3 General error checking

A more general method of error checking is to use 4DML to transform the input into a list of assertions in a high-level programming language of the user's choice. Running the resulting program will then provide the verification. This is usually more concise than writing a separate program that parses the input itself.

For example, if the input contains a matrix of numbers and the sums of all its columns should be equal, then this can be verified by the following program, which is in Python with embedded CML:

```
assert [[cml column between="=="/row between="+"]]
```

This will produce a statement such as `assert 3+4==4+3==2+5`, which is valid in Python. More complex logic, and more detailed error reports, can be obtained by expanding the model.

4.8 Additional methods for writing models

This section describes two additional methods for writing models—using code introspection, and using a graphical interface.

4.8.1 Custom code and code introspection

A 4DML model can be specified as a Python class (Figure 4.31), which is examined by introspection. The class is named after the top-level element of the model, and it contains attributes as member data, and inner classes to specify the child elements. If it has more than one child then the order must be listed separately as introspection does not reveal it, and in some cases it is necessary for classes to specify their real names as attributes if

```
class paragraph:
    nomarkup = 1
    between = "\\par ~\\par\n"
    (before, after) = get_TeX_setup (size = "a4")
    class word:
        between = " "
        begin = "\\mystack"

        class pinyin:
            realname = "language" ; number = 2
            begin = "{" ; end = "}"
            class syllable:
                between = "$\\cdot$"
                def filter(self,text):
                    return make_phonetic_pinyin (text)
        class characters:
            realname = "language" ; number = 1
            begin = "{" ; end = "}"
        class english:
            realname = "language" ; number = 3
            begin = "{" ; end = "}"
            class syllable:
                between = " "

        if english_near_pinyin:
            order = [pinyin, english, characters]
        else: order = [pinyin, characters, english]

def get_TeX_setup(size):
    ... # returns TeX code for begin/end
def make_phonetic_pinyin(text):
    ... # converts pinyin to alternative romanisation
```

Figure 4.31: A 4DML model as a Python class

they are not legal Python identifiers. It is also possible to use other high-level languages similar to Python for this purpose.

Advantages. Besides allowing programmers to use the editors they are familiar with, this approach makes it fairly simple to embed arbitrary code into 4DML models; the code is contained within the methods of the classes. Such methods are able to perform low-level changes to the contents of elements as they are processed by the transformation. For example, code was written to convert romanised Chinese from one romanisation system to another (Section 5.4); this could have been implemented as a large search-and-replace list in 4DML, but it was simpler (for a programmer) to write the code.

Introspection not necessary. Custom code can also be used by CML models—the model specifies the names of filter functions, which are provided separately. Hence it is never *necessary* to use introspection; it is merely an alternative interface should a programmer prefer it.

Rarely used. Custom code is *not* used for structural transformations, which are handled by 4DML itself. It is used only for lower-level transformations on the underlying data

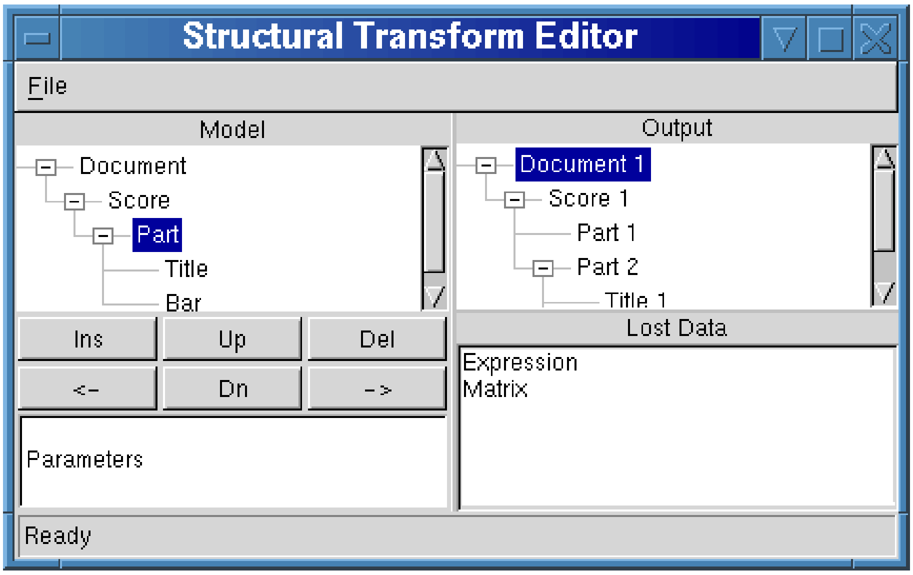


Figure 4.32: A graphical interface to demonstrate 4DML

that lies within the structure. It was not used for any of the examples in this thesis other than the one mentioned above (Chinese romanisation systems).

4.8.2 Graphical interface

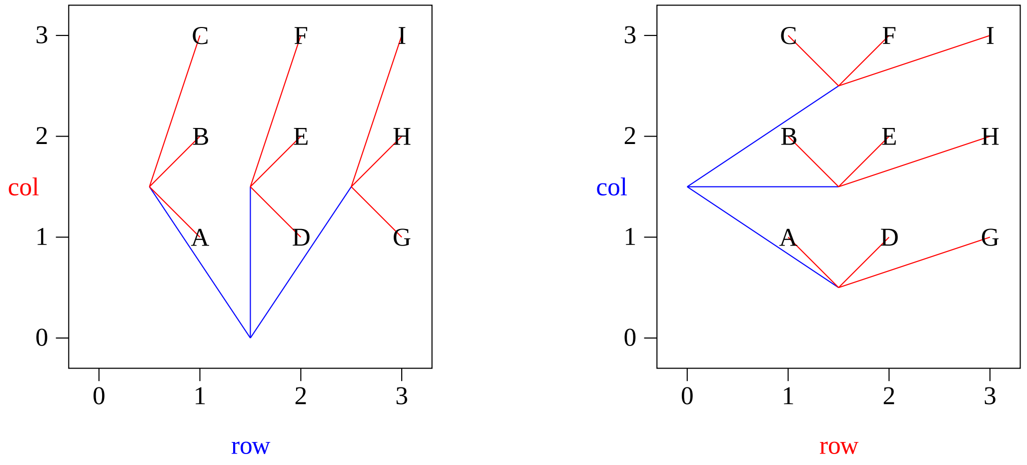
A graphical user interface was produced which uses standard GUI widgets (tree controls) to show the model, the input, the output, and optionally a complete trace of the transformation process and a listing of any data that was not eventually included in the output. The interface allows the model tree to be manipulated and the effects of this are shown in real time (if the computer is fast enough).

Figure 4.32 shows the graphical interface with an artificial example. The controls on the left allow the model to be navigated and modified; the buttons move the highlighted element around the model tree. The views on the right show the transformation's output (again in tree form) and the names of any input elements that are "lost" in the transformation.

Not for production use. This interface is intended to demonstrate the basic principles of 4DML; it is not expected to be used in a production setting. 4DML was designed to have a compact *textual* representation; manipulating it as a graphical tree becomes unwieldy for all but the simplest examples. However, for those who can see it, a graphical representation can be useful for explanatory purposes.

4.9 Alternative ways of understanding 4DML

Thus far the 4DML data structure has been discussed as a four-dimensional pointset. It may also be understood in two other ways, as this section will describe.



(a) Tree interpretation of table from Figure 4.5

(b) Transposing the axes for a different tree

Figure 4.33: A 2-level tree as two dimensions

4.9.1 N-dimensional representation

4DML can be considered as N-dimensional with a varying number of dimensions, one dimension for each level of markup, i.e. for each horizontal layer in Figure 4.4 (b). This concept will be explained by example and then generalised.

Two-dimensional case. Refer back to Figure 4.5. The original data is a two-dimensional object (a table), and the position of each cell within the `row` and `col` markup corresponds directly to its position in the actual two-dimensional space. When the table is represented as a tree with `row` branches above `col` branches, this amounts to interpreting the two spatial dimensions as a two-level tree (Figure 4.33a). The transformation from Figure 4.5 to Figure 4.8 is achieved by treating the axes in a different order when generating the tree (Figure 4.33b). The order in which to treat the axes is given by the model, `col/row` (page 51).

Three-dimensional case. Consider a series of tables, which are to be read in the order: table 1 column 1, table 2 column 1, . . . , table 1 column 2, table 2 column 2, etc. This can be achieved by the model `col/table/row`. The series of tables is a three-dimensional space, the axes of which are re-interpreted by the model—Figure 4.34.

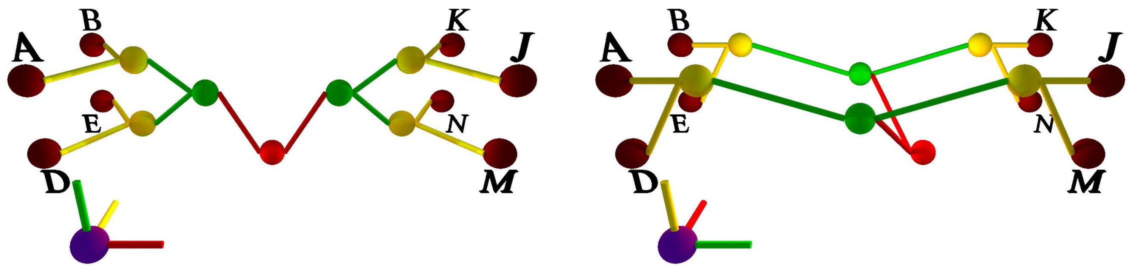
Four-dimensional case. The distributed music example in Figure 4.10 is four-dimensional, the dimensions being *aspect* (pitch or length), part, bar and note. The transformation from Figure 4.10 to Figure 4.11 is essentially a re-ordering of these axes, using the order given by the model (page 51).

Table 1

A	B
D	E

Table 2

J	K
M	N



(a) table/row/col

(b) col/table/row

Figure 4.34: Combining three dimensions into a 3-level tree

N-dimensional case. All 4DML diagrams (such as Figure 4.4) are N-dimensional. The co-ordinates of any given symbol at the bottom of the diagram can be found by reading vertically down the column of markup that is above that symbol. The space can be converted to a tree structure by using the co-ordinates of each symbol as directions from the root to that symbol (a number n means take the nth branch). Many transformations can be expressed by re-ordering the axes; the model's structure is a statement of how the axes should be re-ordered.

A link between two or more points (see Section 4.6.2) can also be represented by placing those points at the same position on a suitable dimension; different dimensions represent different linking systems.

Limitations. This way of thinking is limited because the number of dimensions is not constant throughout the space—some symbols may have more co-ordinates than others—and a single dimension can represent different types of markup in different places. However, the concept is valid in subsets of the 4DML space that represent multi-dimensional matrix-like data—in this case the dimensions in the N-space correspond to the dimensions of the original data.

4.9.2 4DML related to tuple space

The Linda abstraction [32] is a system that allows intelligent agents (autonomous programs designed to assist with reasoning-related tasks) and other applications to share their knowledge. It employs a *tuple space*—an unordered collection of tuples of the form

(type, attribute=value, attribute=value, . . .)

```
(Symbol, data=‘+’, expr=1, equation=1, op=4)
(Symbol, data=‘+’, expr=1, equation=1, op=2)
(Symbol, data=‘=’, equation=1, rel=2)
(Symbol, data=‘3’, equation=1, expr=1, term=3, num=1)
(Symbol, data=‘2’, term=1, exp=2, expr=1, equation=1)
(Symbol, data=‘4’, num=1, term=5, expr=1, equation=1)
(Symbol, data=‘x’, id=1, term=1, equation=1, expr=1)
(Symbol, data=‘x’, term=3, id=2, expr=1, equation=1)
(Symbol, data=‘0’, num=1, term=1, expr=3, equation=1)
```

Figure 4.35: A tuple-space representation of Figure 4.4

where *type* is an indication of the type of information represented by the tuple, and the (attribute, value) pairs, the order of which is not significant, encode any other data that is associated with it.

Representation of notations. Although a tuple space can be used to represent knowledge about the world in general, it can also represent a document, if each tuple corresponds to a symbol in the document and encodes its position and other information. Agents can then perform transformations by accessing the tuple space in an arbitrary order or in parallel, sorting as necessary to ensure that the output is in the correct order.

Reduction to triple space. Any tuple space can be reduced to a triple space—a three-dimensional pointset—by first noting that the “type” field is redundant (since it can be represented as an additional attribute) and then encoding a set of points

(tuple ID, attribute=value)

for each (attribute, value) pair in each tuple, uniquely identifying the tuple that it belongs to with the “tuple ID” field.

Correspondence with 4DML. If the tuples represent symbols in a document and the (attribute, value) pairs represent their positions according to one or more indexing schemes, then the three dimensions of the above-described triple space correspond with three of the four co-ordinates of 4DML. The 4DML unique identification of the symbol that is being marked up will serve as the tuple ID (as well as possibly providing the value of some “contents” attribute), and it serves to connect groups of 4DML points together. The 4DML element name corresponds to the triple space “attribute” field, and the 4DML position number corresponds to the “value”. In other words, each tuple in the tuple space corresponds with a single column in Figure 4.4 (b) with the ordering of the rows being undefined—see Figure 4.35.

The remaining 4DML co-ordinate, depth, makes up for the fact that the triple space (and the tuple space) cannot directly represent hierarchies and recursive structures—these can be stored in the tuple space in an encoded form, but the nature of the tuple space itself does not assist with their representation and auxiliary encoding/decoding would be needed. Because 4DML has a depth co-ordinate, it is able to represent hierarchy and recursion directly, as well as matrix-like or tuple-like structures. Thus 4DML can be considered as a hybrid of a tuple space and a tree-based representation.

4.10 Summary

This chapter discussed the design of the 4DML generalised transformation framework. The framework consists of a generalised markup language for representing structured data, a transformation tool, and two auxiliary languages, Compact Model Language (CML) and Matrix Markup Language (MML). The system allows the programmer to create a *model* of the structure that is to be output, and the transformation is derived from this model.

After a discussion of these, and a description of the parameters available within models, the chapter discussed the facilities for error checking, and then covered other ways of using the system—by code introspection or a graphical interface—and alternative ways of understanding it, as geometric transformations within N-space or as a variation on the tuple space (Section 4.9.2).

The next chapter shows examples of 4DML in use.

5 Example Applications of 4DML

5.1 Introduction

This chapter illustrates some example cases where 4DML can be useful for converting between different notation systems. All of these tasks could have been achieved without the use of 4DML, but normally one would use several different transformational frameworks and tools to achieve the same results, and the code could in some cases be complex and difficult to maintain.

Chapter overview. The chapter shows example transformations from a variety of applications, to demonstrate the generality of 4DML. It begins with a brief 4DML model for extracting data from a complex website, then illustrates transformations between different mathematical and musical notations and demonstrates special-needs foreign-language typesetting. It goes on to show how 4DML can be applied in time management and in the presentation of diagrams, and briefly describes two other potential applications of 4DML that are not directly related to the conversion of notations—error reporting and statistical analysis.

5.2 Website "scraping"

Preamble. For visually-impaired people who live alone, it can be difficult or impossible to see approaching rainclouds, so weather forecasts are a useful aid in making simple decisions such as whether or not to wear a coat. This is especially the case in countries like Britain, which have weather patterns that can change radically every day. Forecasts are available on broadcast media such as radio, but it is necessary to listen at the correct time and (if a local forecast is desired) on the correct frequency, and this is not always practical. Since newspapers are difficult to read and it is costly to obtain forecasts by telephone, an attractive solution (for those who have the connection) is to check the forecasts on Internet websites.

The problem. Internet forecasts are often very detailed, containing data on such things as pressure, temperature, wind speed and pollen counts, over several days and perhaps in several locations (Figure 5.1). For those relying on large print, speech synthesis or Braille, it can take much time to locate the small amount of data that is actually required, particularly if the software cannot guess the most logical way of reading the table (Figure 5.2).

Transformation 1

Application: Allowing a print-disabled person to quickly check the weather forecast

Demonstrates: Brief 4DML model; table transposition

Original notation:	Table of weather reports
Source of input:	Downloaded from BBC website
Input language:	Complex HTML file
Output language:	Plain text
Typeset with:	Any text, speech or Braille terminal
Resulting notation:	Terse weather report

"Scraping". The practice of "scraping" refers to the use of an automatic program to interpret data that was presented with the intention of being read by a human, such as data from a screen display or a complex website. Historically, "scraping" has frequently been used in the area of special-needs access to computing. Its main disadvantage is that any re-design in the layout of the screen or website is likely to break the program that reads it, so these programs need frequent maintenance.

4DML was used as a "website scraping" system to read off appropriate parts of the table in Figure 5.1, and the result was Figure 5.4. The transformation is essentially a matrix transposition and a clipping (a limit on the range of data that is output), and both of these are simple to express in 4DML, leading to a compact transformation "model" that is essentially one line:

```
td start-at=2 end-at=4 between=". "/tr start-at=2 end-at=3 merge/(font, ": ", alt)
```

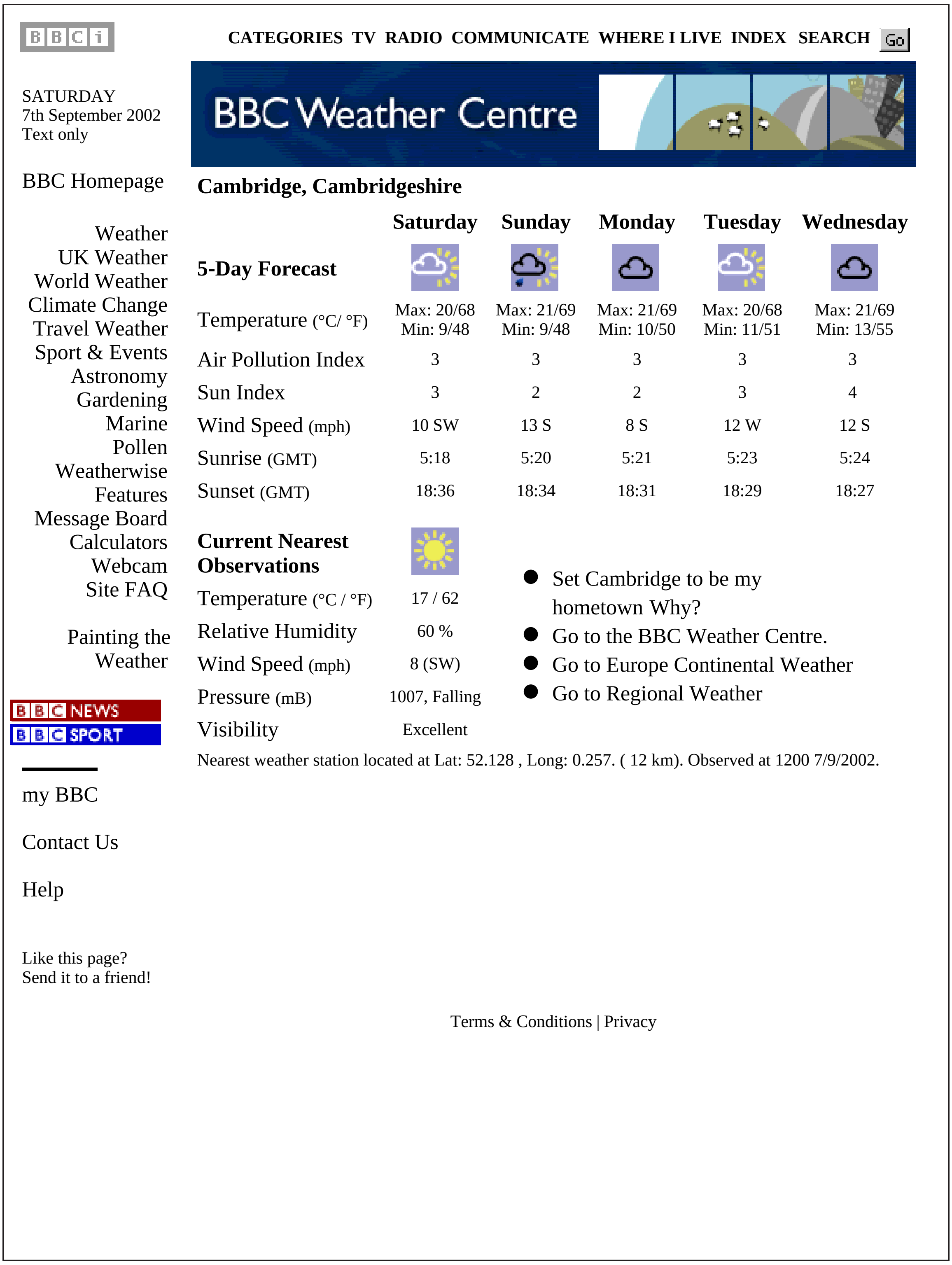
Informally, this means: "For columns (tds) 2 through 4, do the following (while outputting . between each column): For the merged content of rows (trs) 2 and 3, write out the text enclosed by font, then :, then the text enclosed by alt." This happens to be the day name and the weather forecast respectively; they are not explicitly labelled (refer to Figure 5.3). Note that the transposition from columns to rows is implicit in the model's placing of td outside tr. The merging is necessary because the font and alt are in different rows; an alternative approach would be to say:

```
font after=": ", alt
```

or each row could be processed explicitly with:

```
tr number=2/font, ": ", tr number=3/alt
```

This model occasionally needs to be re-written to cope with changes in the site's layout, although not every layout change has affected it. The script has been in daily use for many months.


BBCi
CATEGORIES TV RADIO COMMUNICATE WHERE I LIVE INDEX SEARCH Go
BBC Weather Centre

SATURDAY
7th September 2002
Text only

Cambridge, Cambridgeshire

	Saturday	**Sunday**	**Monday**	**Tuesday**	**Wednesday**
5-Day Forecast					
Temperature (°C/ °F)	Max: 20/68 Min: 9/48	Max: 21/69 Min: 9/48	Max: 21/69 Min: 10/50	Max: 20/68 Min: 11/51	Max: 21/69 Min: 13/55
Air Pollution Index	3	3	3	3	3
Sun Index	3	2	2	3	4
Wind Speed (mph)	10 SW	13 S	8 S	12 W	12 S
Sunrise (GMT)	5:18	5:20	5:21	5:23	5:24
Sunset (GMT)	18:36	18:34	18:31	18:29	18:27

Current Nearest Observations	
Temperature (°C / °F)	17 / 62
Relative Humidity	60 %
Wind Speed (mph)	8 (SW)
Pressure (mB)	1007, Falling
Visibility	Excellent

- Set Cambridge to be my hometown Why?
- Go to the BBC Weather Centre.
- Go to Europe Continental Weather
- Go to Regional Weather

Nearest weather station located at Lat: 52.128 , Long: 0.257. (12 km). Observed at 1200 7/9/2002.

Like this page?
Send it to a friend!

Terms & Conditions | Privacy

Figure 5.1: Internet forecasts can be detailed (note that the “text only” link takes you to a page resembling Figure 5.2)

Speech synthesizer: "Cambridge, Cambridgeshire Saturday Sunday Monday Tuesday Wednesday 5-day forecast sunny intervals light showers cloudy sunny intervals cloudy temperature degrees C degrees F max 20 68 min 9 48 max 21 69 min 9 48 max 21 69 min 10 50 max 20 68 min 11 51 max 21 69 min 13 55 air pollution index 3 3 3 3 3 sun index 3 2 2 3 4 wind speed MPH 10 SW 13 S 8 S 12 W 12 S sunrise GMT 5 18 5 20 5 21 5 23 5 24 sunset GMT 18 36 18 34 18 31 18 29 18 27 current nearest observations..."

Figure 5.2: Locating information can be difficult if the software cannot guess the logical reading order

If the website were to release weather data in a standardised format that is specifically intended for processing by a program, then this would remove the need to re-write the model whenever the web designers change the layout. However, it is often the case (at the time of writing) that data in such formats is only available for a fee, since it would make it easier to set up a competing source of weather forecasts.

5.3 Mathematics reading

Mathematics to speech. 4DML was used to parse an electronic document that contained mathematical notation, and to output the result as text suitable for a speech synthesiser, rendering the mathematical expressions appropriately. For example, the expression

$$\sum_{n=0}^{k} \frac{f^n a}{n}$$

(source shown in Figure 5.5) became "sigma from n equals 0 to k of f to the n a over n".

The model for this is somewhat larger because of the number of different mathematical symbols it needs to translate. A partial version of it is shown in Figure 5.6 in XML format. The model does not need to do any unusual transpositions; instead it uses 4DML's `sequential` parameter to step through the MathML sequentially using top-down recursion, in imitation of other transformation frameworks such as XSLT. Each item in the structure will be matched against some substitution rules provided in the model, such as

```
<mo value="&#x2200;"> for all </mo>
```

meaning "find any `mo` (operator) elements that contain the entity code `&#x2200;`, and output `for all` for any that are found"—as the data is being processed sequentially, one item at a time, this will find 0 or 1 elements. Another example is:

```
<mfrac>
  <anyName number="1" call="math" sequential="1" />
 over
  <anyName number="2" call="math" sequential="1" />
</mfrac>
```

```
<tr>
  <td colspan="6"><font size="2">Cambridge</font></td>
</tr>
<tr>
  <td>&nbsp;</td>
  <td><font size="2">Saturday</font></td>
  <td><font size="2">Sunday</font></td>
  <td><font size="2">Monday</font></td>
  <td><font size="2">Tuesday</font></td>
  <td><font size="2">Wednesday</font></td>
</tr>
<tr>
  <td><font size="2">5-Day Forecast</font></td>
  <td><img src="2.gif" alt="sunny intervals"></td>
  <td><img src="12.gif" alt="light showers"></td>
  <td><img src="8.gif" alt="cloudy"></td>
  <td><img src="12.gif" alt="light showers"></td>
  <td><img src="8.gif" alt="cloudy"></td>
</tr>
<tr>
  <td><font face="Arial,Helvetica" size="2">Temperature
  ...
```

Key: — target data
— **markup identified by the model**
— other data and markup

Note: HTML has been simplified for illustrative purposes

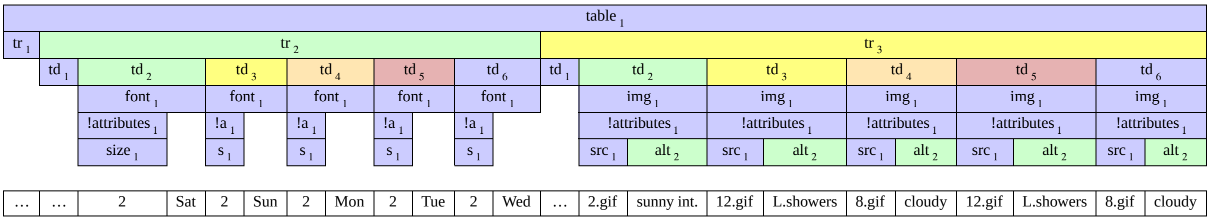


Figure 5.3: Fragment of HTML source for Figure 5.1, and 4DML representation

SPEECH SYNTHESIZER: "Saturday: sunny intervals. Sunday: light showers. Monday: cloudy"

Figure 5.4: Result of website "scraping"

Transformation 2

Application: Assisting a totally blind person in reading mathematics

Demonstrates: Top-down sequential processing; XML model

Original notation:	Mathematics
Source of input:	TeX to MathML converter (tex4ht)
Input language:	MathML
Output language:	Plain text
Typeset with:	Any text, speech or Braille terminal
Resulting notation:	English reading of the mathematics

Transformation 3

Application: Illustrating to sighted people the workings of a Braille mathematics code

Demonstrates: Different types of output from the same input

Original notation:	Mathematics
Source of input:	TeX to MathML converter (tex4ht)
Input language:	MathML
Output language:	The language of the typesetter LaTeX
Typeset with:	LaTeX
Resulting notation:	Annotated diagram of Braille mathematics

(a) Original LaTeX file

```
\documentclass{article}
\begin{document}
$\sum_{n=0}^{k} \frac{f^{n}a}{n}$
\end{document}
```

(b) MathML generated using the TeX tool TeX4ht

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="mathml.css"?>
<?xml-stylesheet type="text/css" href="matheg.css"?>
<!DOCTYPE html SYSTEM "mathml.dtd"
[ <!ENTITY mmlns "http://www.w3.org/1998/Math/MathML"> ]>
<html xmlns:math="http://www.w3.org/1998/Math/MathML"
xmlns="http://www.w3.org/TR/REC-html40">
  <head>
    <title>matheg.xml</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
    <link rel="stylesheet" type="text/css" href="matheg.css" />
  </head>
  <body> <!--l. 3-->
    <math xmlns="&mmlns;" mode="inline">
      <msubsup>
        <mo>&#x2211;</mo>
        <mrow> <mi>n</mi> <mo>=</mo> <mn>0</mn> </mrow>
        <mrow> <mi>k</mi> </mrow>
      </msubsup>
      <mfrac>
        <mrow>
          <msup>
            <mi>f</mi>
            <mrow> <mi>n</mi> </mrow>
          </msup>
          <mi>a</mi>
        </mrow>
        <mrow> <mi>n</mi> </mrow>
      </mfrac>
    </math>
 </body>
</html>
```

Figure 5.5: Mathematics in a LaTeX file, and the MathML equivalent

which means "for any `mfrac` (fraction) elements, process the first child (using `call="math"` to recursively nest the model within itself), output `over` and process the second child".

Mathematics to Braille. By changing the model, Braille output can be produced (Figure 5.7), in this case using the Nemeth code [20] but it is possible to use other codes. Here the Braille is in the form of annotated TEX graphics, but it can also be output as codes suitable for controlling an automated Braille embosser or a Braille display.

Spoken ambiguity. Spoken mathematics is ambiguous—one could interpret "f to the n a over n" as $f^n a/n$, f^{na}/n, or the unlikely $f^{na/n}$. This ambiguity could be resolved by being more specific in the speech, but it is not always clear (from the standpoint of automation) precisely how specific the speech should be—*too* specific would be unwieldy, as in "begin group, f, begin superscript, n end superscript, a, end group, over n". Normally mathematicians and scientists rely on context and "common sense" to resolve ambiguities and to decide how explicit or otherwise their speech should be, depending on the needs of the case at hand. In the case of computer speech synthesis, this judgement is difficult to achieve unless it has somehow been encoded into the document beforehand (which is rarely done).

AsTeR (mentioned on page 17) works around this problem by employing other audio dimensions such as changing the characteristics of the computer's voice and inserting non-speech audio, and it also allows the user to browse the document interactively and choose between several different audio renderings (perhaps designed by the user). 4DML could of course be used to emulate AsTeR's transformations, provided that a suitable output device is available (not all speech synthesisers are capable of the level of control that AsTeR requires). However, ambiguity is not always an issue if all that is required is an overview of a document that contains relatively straightforward mathematics.

Written ambiguity. Occasionally, one printed symbol maps to several different Braille codes depending on the context. For example, ! could be a factorial (⠯) or a literary exclamation mark (⠖), and $|A|$ could mean A's absolute value (⠳) or its determinant (⠠⠳). A human who is reading printed mathematics can usually judge, but there is no difference in languages like TEX, which was designed for typesetting printed mathematical notation and therefore has no reason to represent expressions any more unambiguously than is required for that purpose. This can result in incorrect Braille; the Braille reader will then have to be aware of how mathematics is written in print in order to understand what happened. This is why many establishments for the blind are unwilling to distribute automatic transcription software unless its output will be proof-read before being given to students.

This ambiguity can be resolved if the electronic document is encoded in an appropriate markup language that contains all the necessary information. This could require manual intervention.

```
 <?xml version="1.0"?>
<body sequential="1"> (process items sequentially; don't regroup them)
  <div sequential="1" rename="" call="body"/> (treat 'div' like 'body')
  <math sequential="1" wildcard="anyName" nomarkup="1" between=", ">
      <mo value="&#x003C;"> less than </mo>
      <mo value="="> equals </mo>
      <mo value="&#x003E;"> greater than </mo>
      <mo value="&#x2200;"> for all </mo>
      <mo value="&#x2203;"> there exists </mo>
. . . more substitutions of MathML/Unicode symbols follow
. . .
      <mn/> (keep numbers as they are)
      <mrow sequential="1" call="math"/> (treat 'mrow' as 'math')
      <mfrac> what fractions should look like:
          <anyName number="1" call="math" sequential="1" />
over
          <anyName number="2" call="math" sequential="1" />
      </mfrac>
      <msub> what subscripts should look like:
          <anyName number="1" call="math" sequential="1" />
sub
          <anyName number="2" call="math" sequential="1" />
      </msub>
. . . (and so on)
  </math>
  <p sequential="cdata"> (write out ordinary text; check for 'math' inside it)
      <span sequential="cdata" rename="" call="p"/>
      <math call="math" sequential="1" wildcard="anyName" nomarkup="1" between=", " after=" "/>
  </p>
</body>
```

Figure 5.6: Part of 4DML model for transforming mathematics to speech

$$\sum_{n=0}^{k} \frac{f^n a}{n}$$

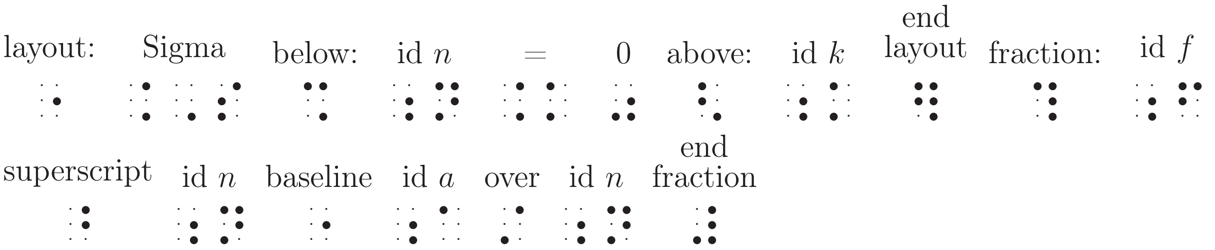


Figure 5.7: Annotated Braille mathematics (Nemeth linear code)

Transformation 4

Application: Assisting Western beginners (particularly print-disabled ones) with learning Chinese

Demonstrates: Customisable output; ability to use different input languages; mixing 4DML with Python; different types of output from the same input

Original notation:	Chinese text
Source of input:	Text editor with Chinese dictionary
Input language:	Dictionary data in various layouts according to what is convenient
Output language:	The language of the typesetter LaTeX
Typeset with:	CJK-LaTeX
Resulting notation:	Customised presentation of Chinese text

duìbùqì wǒ shìtú shuō zhōngwén kěshì wǒ shuō de bùhǎo yīncǐ wǒde xìn dàduō yīngwén shūxiě le Rúguǒ
对不起、我 试图 说 中文 可是 我 说 的 不好、因此 我的 信 大多 英文 书写 了。如果
jīntiānde tiānqì hǎo
今天的 天气 好、etc

Figure 5.8: The “ruby” system of writing pinyin above Chinese characters

5.4 Typesetting for special-needs language learning

Preamble. When teaching written Chinese to Western students, Chinese characters are often written with small pronunciation guides around them, and sometimes the meaning of each character or group of characters is written alongside them in the language of the student.

There are several systems for indicating pronunciation; some use special phonetic symbols (such as *zhuyin*, which is informally known as bopomofu), and others employ the Latin alphabet. These *pinyin* (alphabetical) systems use either numbers or special accents to indicate tonal inflection, and they use various different spellings that are not always intuitive to English people; this is a common source of bad pronunciation. Different students prefer different *pinyin* systems, and many beginners prefer to devise their own systems for private use, but there are advantages in learning one of the standardised systems (such as *Hanyu Pinyin* which is used in mainland China) since it is widely used in printed books, dictionaries, and computer software.

“Weaning”. Many students try to wean themselves from their private notations onto the standard that they wish to learn. Such weaning is usually done by writing both notations in parallel and progressively deleting parts of the private notation according to the student’s progress. In this way students can be weaned from one *pinyin* system to another, and also towards reading the Chinese characters directly.

Difficulties for students with low vision. The “ruby” system of writing pinyin above the characters (Figure 5.8) is difficult for partially-sighted students because it employs very small print, and if it is enlarged without adjustment then the result can be unwieldy and give rise to *tracking* problems, where people with certain sight conditions lose track of which line they are reading. Additionally, if these students want to produce a customised version of the notation for themselves then they are impeded by the graphical way in which the symbols are positioned relative to each other; they either have to write it by hand or use a graphical wordprocessor, and both of these things can be difficult for visually-impaired students.

Customised output. 4DML allows a user to produce a model of the desired output in the language of a Chinese-capable, text-driven typesetting package such as CJK–LaTeX. In Figure 5.11, the different lines of text, which have been printed at similar sizes to facilitate zooming, have been brought very close together so as to aid tracking, and colour has been used to compensate for the crowding that this gives and to further assist with tracking (shades of grey can also be effective). The blue private notation (which has been generated automatically) has mostly been taken away, and the few Chinese characters that the user has learned have been duplicated into the pinyin line. Syllable separation dots (which are not standard in Chinese) have also been used, as have parentheses to indicate a higher level of grouping.

The CJK–LaTeX (Figure 5.9) is fairly complex due to the customised kerning (bringing text close together); the model (which was shown in part in Figure 4.31 on page 70) is an example of the use of a Python class (Section 4.8.1) to mix 4DML structural transformation with arbitrary lower-level code, in this case the code to convert romanised Chinese from one romanisation system to another. The Python class hides the complexity of the CJK–LaTeX.

By making small adjustments to the model, many different variations of this notation have been produced. Such adjustments include:

1. changing the fonts, spacing and colours;
2. changing the order in which the lines appear, or deleting some lines altogether;
3. controlling which Chinese characters and which aspects of pinyin are assumed to be known;
4. adding symbols to denote musical tones for songs, and presentation techniques (e.g. gestures).

These adjustments allow for variations in:

1. sight and reading characteristics, both between students and over time,
2. display technology—besides colour and greyscale printouts, the notation has been used in monochrome on a 10cm×4cm PDA screen, where large print means only one line is visible,
3. the task—for example, reading a script to an audience can require much larger text, whereas collaborating with a teacher requires smaller text with less markup and space for handwritten corrections.

Seperate models were written to produce the following additional formats from the same input:

- Conventional “ruby” notation (Figure 5.8)
- A plain text version of the notation for a Chinese-capable email client;
- Western musical staff notation for the songs;
- Chinese Braille.

```
\documentclass[12pt,a5paper,landscape]{article}
\usepackage{CJK} \usepackage{pinyin} \usepackage{color}
\topmargin -5mm \marginparwidth 0in \oddsidemargin 5mm \evensidemargin 5mm
\textheight 138mm \textwidth 200mm \makeatletter \makeatother

\def\vkern#1#2#3#4{\raisebox{0pt}[#1][#2]{\textcolor{#3}{#4}}}
% (kern #4 to #1 high and #2 deep, colour #3)

\def\mystack#1#2#3{ \huge
  \shortstack{
    #1 \\ % pronunciation + pinyin
    \vkern{1ex}{0.5ex}{green}{#2} \\ % characters
    \vkern{1.5ex}{2ex}{black}{#3} % english
  }
}
\def\flat#1{\Huge\vkern{1.6ex}{0.5ex}{black}{#1}} % pinyin without help
\def\help#1#2{\shortstack{    % pinyin with help:
  \vkern{1.3ex}{0ex}{blue}{\LARGE #2} \\ % pronunciation
  \flat{#1}                              % pinyin
}} \def\red#1{\textcolor{red}{#1}}

\begin{document}\pagestyle{empty}
\begin{CJK*}{GB}{song}
\raggedright
\mystack{\flat{\dui4}$\cdot$\flat{不}$\cdot$\flat{\qi4、}}{I'm sorry,}{对不起、}
\mystack{\flat{\red(\Wo3}}{I}{我}
\mystack{\help{\shi4}{shr}$\cdot$\flat{\tu2}}{attempt}{试图}
\mystack{\flat{说}}{speak}{说}
\mystack{\flat{中}$\cdot$\flat{\wen2\red)}}{Chinese}{中文}
\mystack{\flat{\ke3}$\cdot$\help{\shi4}{shr}}{but}{可是}
\mystack{\flat{\red(\wo3}}{I}{我}
\mystack{\flat{说}}{speak}{说}
\mystack{\flat{的}}{}{的}
\mystack{\flat{不}$\cdot$\flat{\hao3\red)、}}{no good,}{不好、}
\mystack{\flat{\yin1}$\cdot$\help{\ci3}{tsz}}{therefore}{因此}
\mystack{\flat{\red(\wo3}$\cdot$\flat{的}}{my}{我的}
\mystack{\flat{\xin4\red)}}{letter}{信}
\mystack{\flat{\da4}$\cdot$\help{\duo1}{doar}}{mostly}{大多}
\mystack{\flat{\red(\ying1}$\cdot$\flat{\wen2}}{English}{英文}
\mystack{\flat{\shu1}$\cdot$\flat{\xie3}}{write}{书写}
\mystack{\flat{了\red).}}{}{了。}
\mystack{\flat{\Ru2}$\cdot$\help{\guo3}{goar}}{If}{如果}
\mystack{\flat{\red(\jin1}$\cdot$\flat{\tian1}$\cdot$\flat{的}}{today's}{今天的}
\mystack{\flat{\tian1}$\cdot$\flat{\qi4}}{weather}{天气}
\mystack{\flat{\hao3\red)、}}{fine,}{好}
\mystack{}{}{etc}\par
\end{CJK*}\end{document}
```

Figure 5.9: CJK–LaTeX for Figure 5.11

Input. There are many established ways of inputting ideographic characters; most of these involve typing a character's pronunciation (in some dialect) or codes that give clues about its appearance, and then selecting from the characters that match those criteria. This essentially amounts to making queries on a database or *dictionary* of characters, and for the language student the system can be extended as follows:

1. Use a dictionary that includes definitions in English or another language, and allow searches on these definitions *as well as* any of the character's other attributes (using this method it is also possible to retrieve a sequence of several characters in one operation);
2. Once the character(s) have been identified, insert into the document the entire dictionary entry (including pronunciation and definition), not just the character itself.

This means that the input to the 4DML transformation is a sequence of dictionary entries, so that 4DML has all the necessary information to produce annotated output. Figure 5.10a is in the same format as the CEDICT dictionary [23]—its lines are copied directly from CEDICT with some editing for readability. This can be done either manually or with a specially-programmed input method, such as the one this author implemented for Emacs.

Other input formats. If Chinese characters have already been provided then it can be unclear how they should be grouped into words (which can be formed from two or more characters), and in this case it is useful to keep the pinyin together so that its grouping can be changed quickly, for example by replacing a space with a hyphen—Figure 5.10b. Sometimes Chinese characters may not be available at all, and only pinyin and English (or another language) is present.

If the input has been provided by others then it may be in a more esoteric format, such as song lyrics (Figure 5.10c), where a line of pinyin is placed above a line of characters, and pairs of lines from different stanzas are interleaved.

Usage. The system has thus far been used to typeset 147 documents to assist learners of Chinese; these include 5 presentations and 71 Chinese songs. Participants had a wide range of competence levels and sensory abilities.

5.5 Processing music

4DML was used for several music-related transformations, including "distributed" or aspect-oriented music encoding, aspect-oriented composition, and typesetting Japanese koto notation.

5.5.1 Distributed music encoding

Preamble. Handwritten music can be difficult to read at speed. Unless the scribe has been meticulous, working with it requires good visual acuity and recognition skills, which not every musician has all of the time. Good-quality printed music can help, especially if

(a) CEDICT format:

```
我 [Wo3] /I/
试图 [shi4 tu2] /attempt/
说 [shuo1] /speak/
中文 [zhong1 wen2] /Chinese/
```

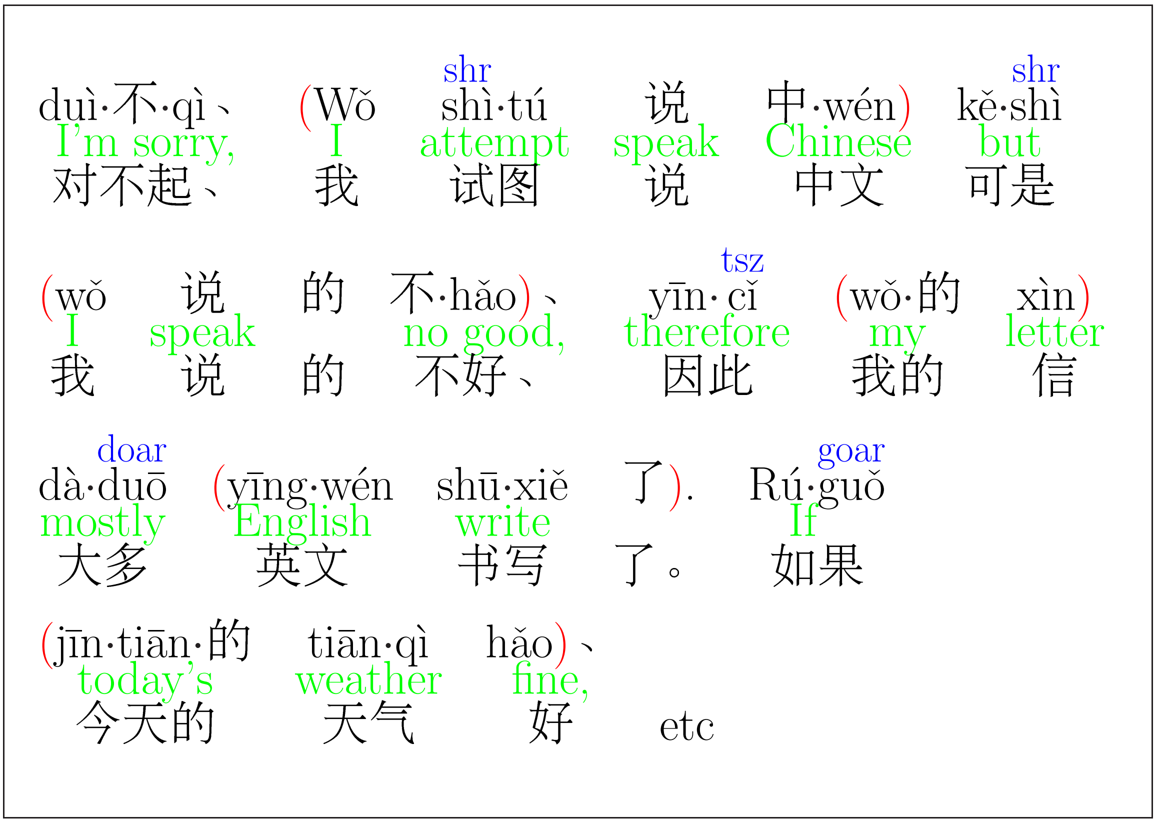
(b) PinYin-based format:

```
Wo3 shi4-tu2=attempt shuo1 zhong1-wen2
```

(c) Song lyrics format:

```
Shang4 di4 xian4 yi3 xia4 ling4, pai4 qian3
上 帝 现 已 下 令, 派 遣
Shang4 di4 di4 shang4 zi3 min2, wang2 guo2
上 帝 地 上 子 民, 王 国

guang1 zhao4 lin2 di4 shang4 ...
光 照 临 地 上 ...
qin2 xuan1 jiang3; an1 wei4 ...
勤 宣 讲; 安 慰 ...
```

Figure 5.10: Different ways of inputting annotated Chinese



		shr			shr
duì·不·qì、	(Wǒ	shì·tú	说	中·wén)	kě·shì
I'm sorry,	I	attempt	speak	Chinese	but
对不起、	我	试图	说	中文	可是

				tsz		
(wǒ	说	的	不·hǎo)、	yīn· cǐ	(wǒ·的	xìn)
I	speak		no good,	therefore	my	letter
我	说	的	不好、	因此	我的	信

doar				goar
dà·duō	(yīng·wén	shū·xiě	了).	Rú·guǒ
mostly	English	write		If
大多	英文	书写	了。	如果

(jīn·tiān·的	tiān·qì	hǎo)、	
today's	weather	fine,	
今天的	天气	好	etc

Figure 5.11: A customised notation to assist with Chinese studies

Transformation 5

Application: Allowing someone to quickly encode music while not being fluent with the encoding language, and allowing several people with different skills to simultaneously work on a music encoding project

Demonstrates: Aspect-oriented input

Original notation:	Hand-written Western musical staff
Source of input:	Text editor
Input language:	An invented music language designed for quick input of that writing style
Output language:	The music language of M-Tx
Typeset with:	M-Tx, PMX and MusiXTeX
Resulting notation:	Printed Western musical staff

Transformation 6

Application: Assisting a print-disabled composer

Demonstrates: Aspect-oriented input; input language invented for the task; use of more than one typesetting system

Original notation:	None (composition)
Source of input:	Text editor on PDA
Input language:	An invented music language designed for the composition
Output language:	The music language of GNU Lilypond
Typeset with:	GNU Lilypond
Resulting notation:	Printed Western musical staff

- Note letters
- Octaves
- Enharmonics
- Note values
- Dots
- Tuplets
- Phrasing
- Articulation
- Ornaments
- Dynamics
- Text
- Time and key changes
- Typographic adjustments

Figure 5.12: Some different aspects of Western musical notation

it is possible to adjust the print size without needing unwieldy large photocopies. But it is necessary first to input the music into a computer.

There are essentially four ways of inputting Western musical notation into a computer:

1. Scanning. This is only feasible if the music is of sufficiently good quality to begin with, and is rarely possible with handwritten music.
2. Using a *direct manipulation* music publishing system such as *Sibelius* [24]. Such software can be difficult to use for people with print disabilities.
3. Playing the music on a keyboard or similar. Due to the limitations in artificial "aural skills", the resulting notation is usually inaccurate for all but the simplest music.
4. Writing in a "musical code" (also called "little music language")—a special computer language that gives instructions to a music typesetting program.

Since the latter option is the most feasible for print-disabled people, that is the one that is examined here.

The problem. Modern Western musical notation has a large "vocabulary" of possible symbols (Figure 5.12), so the computer languages that represent it have to be fairly complex. This means that those who wish to write in such languages will either have to spend time learning them (which can be too much for an occasional user), or will be slowed down by the frequent need to refer to a reference manual or online help system. If the user is also the composer then this problem might restrict the composition.

Distributed music encoding. The author's approach to addressing this problem is to go through the piece several times, each time encoding just one or two of the aspects in Figure 5.12. Thus the user can type in *all* of the note letters in the piece, then go back to the beginning and type in *all* of the octaves, and so on. This is more efficient because only a small amount of vocabulary needs to be considered at any one time. It is called *distributed* music encoding because it also introduces the possibility of distributing the work-load between several people with limited training (one person could be trained just to type in the pitches, another to type in the note-values, another to type in the accidentals, and so on).

Explanation of Figure 5.13. This figure is an extract from some musical data in distributed music encoding. The musical aspects that are visible are:

1. Pitches, here written as note letters; `r` denotes a rest (silence).
2. Accidentals, here written as `s` for sharp (♯), `f` for flat (♭) and `n` for natural (♮), with `.` to skip over a note that does not have an accidental—this is only necessary if there is a note later in that bar which has an accidental. It is not necessary to "pad out" bars by skipping over the trailing notes, since whitespace will move to the next bar.
3. Octave changes: `+` increases the octave and `-` decreases it; the sign can be repeated for changes of two or more octaves. Comma (`,`) is used to advance to the next note—since an octave change may take more than one character, it is not possible to use `.` in the same way as with the accidentals. However, again it is not necessary to "pad out" bars—when at least one character has been written, the bar can be ended at any time without needing a `,` for each remaining note. If the bar separator were something other than `whitespace` then it would not even be necessary to write one character.

   It is also possible to write absolute octave numbers rather than relative changes.
4. Durations: `0`=semibreve (whole note), `2`=minim (half note) `4`=crotchet (quarter note), `8`=quaver (eighth note), `1`=semiquaver (sixteenth note).

In each case, the letters, digits and punctuation characters that are used may be changed if desired (for example, if the user is not from an English background and does not spell "sharp" with an S). Other aspects of the music (such as ornaments and dynamics) are encoded separately later.

Process. The input (Figure 5.13) can be consistency-checked using the techniques described in Section 4.7; it is then transformed using the model in Figure 5.14, interleaving the different aspects of music in the manner described in Section 4.3.1, to give the language of the music typesetter M-Tx (Figure 5.15) which is then typeset to produce Figure 5.16.

Aspect-oriented composition. Some modern styles of composition can also lend themselves to a distributed or *aspect-oriented* construction, as shown in Figures 5.17 and 5.18. The independent aspects of the notation that are being added progressively are not necessarily the aspects of Figure 5.12; rather, they are the aspects of the compositional framework defined by the composer (in this case including such things as "time distortion"—irregular short-term variation of the speed), which is then converted into standard musical notation by the 4DML model. The idea is similar to the separation of concerns in aspect-oriented programming [19].

A different music typesetting program was used (GNU Lilypond), showing that 4DML does not rely on any particular music program.

```
begin music
begin part

!block pitch
have whitespace character as bar note

r rrrd ddddfca aarrd ddddfca aadce gfcdfeca gfcddfeeg
gfbagffg dcfffg feaabb ddcc bbaa gfgaabaadfa ddcbbdf
baggbdgfe egfee gfeebdrad daffaaaad drdcbbdf baggbdgfe egfee
gfeebdrad daffaaaac ddddbdd ddddbdd rgfr rgfr dfca ddgfc
agfr gfddgfc cgfdd dgfccgf ddgfa abrgrf rdd drrrrd ddddfca
aarrd ddddfca aadce gfcdfeca gfcddfeeg gfbagffg dcfffg
feaabb ddcc bbaa gfdbgf cagfbg gegfbggegfgfbgge gfgfdb
gfcagf bggegfbggegfgfbg gegfgfgfd daadfadr r rrgeb rbgrg
frrr rf fadd d
!endblock

!block accidental
have whitespace character . as bar note rubbish

. . .....s . .....s ...s ..s ..s ......s .sn.s .....n ..n nn
.s...n...n . .f......f ...f ..f . . .f......f ...f ..f
. . . . . ..s ....s . ......s . ...s . . . . .....s . .....s
...s ..s ..s ......s .sn.s s....n ..n nn ...f s . . ..s
!endblock

!block octave
have whitespace , as bar note

, ,,,++ ,-,+,- ,,,,+
,-,+,- , , + , , , ,-,+ ,-,+ ,,,,,,-
,,,,,- ,,,,- , ,
,,+,,-,+,,-,+ ,,,,,,- ,,,,- ,
, ,,+,,-,+,,-,+ ,-,+
,-,+ ,- , + , , ,,+,,-,,+ ,,,+
, +,,- , ,+ , ,-,+,- ,,,,+ ,-,+,-
, , + , , , ,-,+
,-,+ ,,+ +
, ,,,,+ ,,+
, ,,,,,,,,+ ,- ,
,,-,,+ , , , ,,-,+
!endblock

!block duration
have whitespace character . as bar note rubbish

0 2488 8881144 28888
8881144 28114 11481148
114111148 11481148 114848
114848 4882 4882 88888883333
48114.. 8114..811 48114 8112..5..
4..4..4.. 888114.. 8114..811 48114
```

Figure 5.13: Part of a file written using distributed music encoding

```
Title: Sad Fountains
Composer: C. J. Brown
Name: Flute
Flats: 1
Style: solo
Systems: 18
Pages: 2
Space: 0 0
Meter: 4/4

[[cml music flatten/
  bar expected between="&#10;&#10;" /
   part expected between="&#10;" / (
    uptext begin="U: " end="&#10;",
    keychg/posn number=1 after=" ",
    note between=" " / (
      tie/posn number=1 after=" ",
      pitch,
      accidental,
      dot,
      duration,
      octave,
      shift,
      tuplet value="3" begin="x",
      tuplet value="6" begin="x",
      artic value="." begin=" o",
      artic value="_" begin=" o",
      artic value=">" begin=" o",
      artic value="f" begin=" o",
      tie/posn start-at=2 begin=" ",
      dynam begin=" "
     ),
    keychg/posn number=2 before=" "
   ) ]]
```

Figure 5.14: A 4DML model in M-Tx format (the embedded code reads the data from Figure 5.13)

```
Title: Sad Fountains
Composer: C. J. Brown
Name: Flute
Flats: 1
Style: solo
Systems: 18
Pages: 2
Space: 0 0
Meter: 4/4

U: @+7 Adagietto
  r0

U: ~ ~ ~ pp
  r2   r4   r8  (t d8++ o_ Dpp

U:
  d8 )t  ( d8-   d8+ o_ ) ( d1-  f1 )  cs4 o_ (1t a4

U:
 (2t a2 )1t   a8 )2t   r8 (t r8 d8+ o_

U:
  d8 )t  ( d8-   d8+ o_ ) ( d1-  f1 )  cs4 o_ (1t a4

U: ~ ~ ~ ~ poco~espress
 (2t a2 )1t   a8 )2t D< d1  cs1 D< e4 o_

U:
 ( g1   f1   cs4   d8 o. )  ( f1   e1   c4 D<  a8 ) D<

U:
 ( g1+   f1   cs4   d1 )   d1  ( f1   e1 D<  e4   g8 o. ) D<


U:
 ( g1   f1   b4   a8 o. )  ( g1   f1   fs4 D<  g8 o. ) D<

U:
 ( d1   cs1  (2+0+2 fnd4  f8 )2+0-1  fs4 ) g8 o.

U:
 ( f1   e1  (2+0+2 ad4   a8 )2+0-1 D<  b4 )  bn8 o. D<

U: f
  d4 Df  d8- (t cn8+  c2 o_ )t

U:
  bn4   bn8-  (t a8+  a2 o_ )t

U:
 ( g8   fs8   g8 (t a8  a8 )t  bn8 )  a8- ( a3  d3  fn3  a3
```

Figure 5.15: M-Tx code automatically generated from Figure 5.14

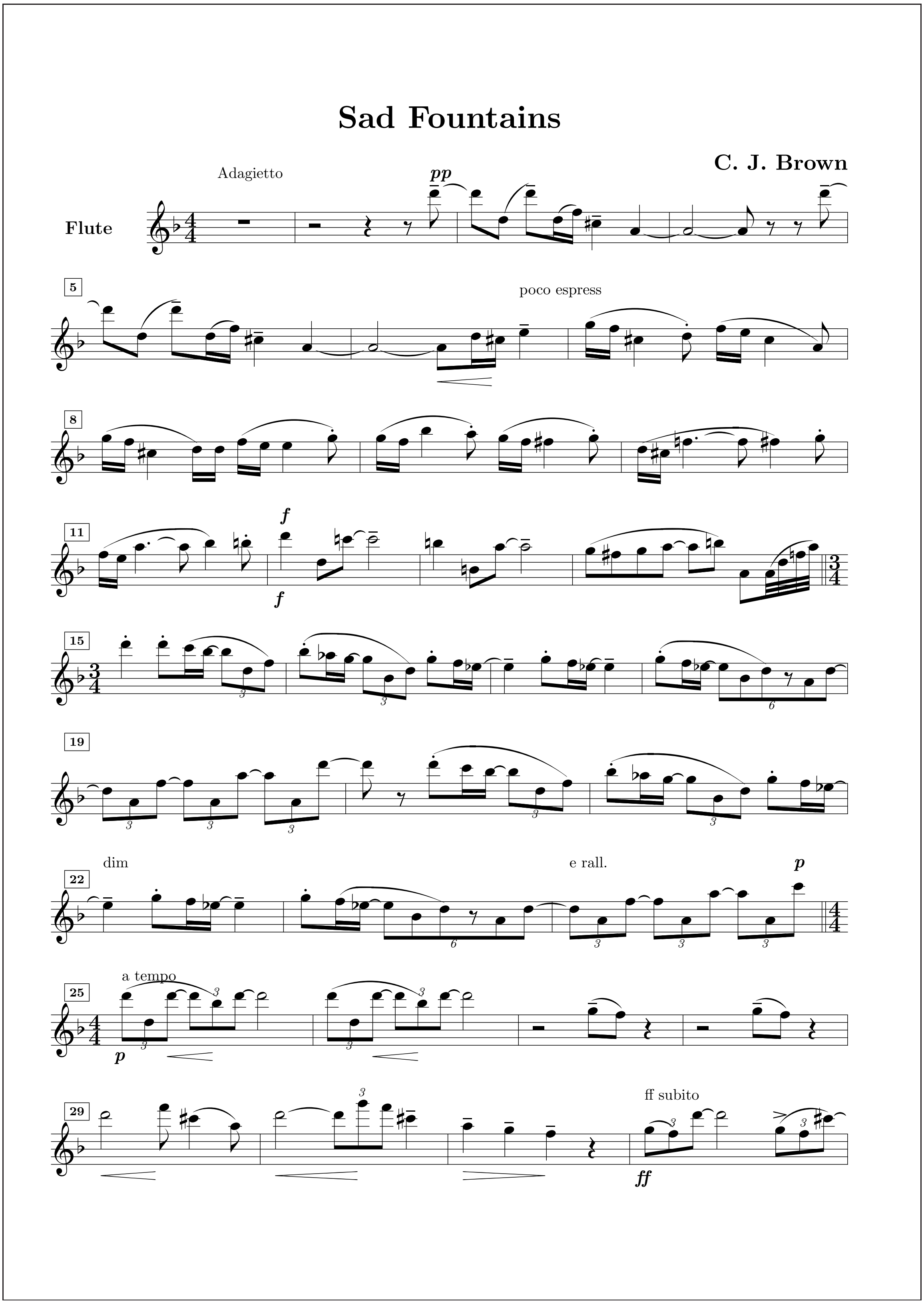


Figure 5.16: The result of processing Figure 5.14

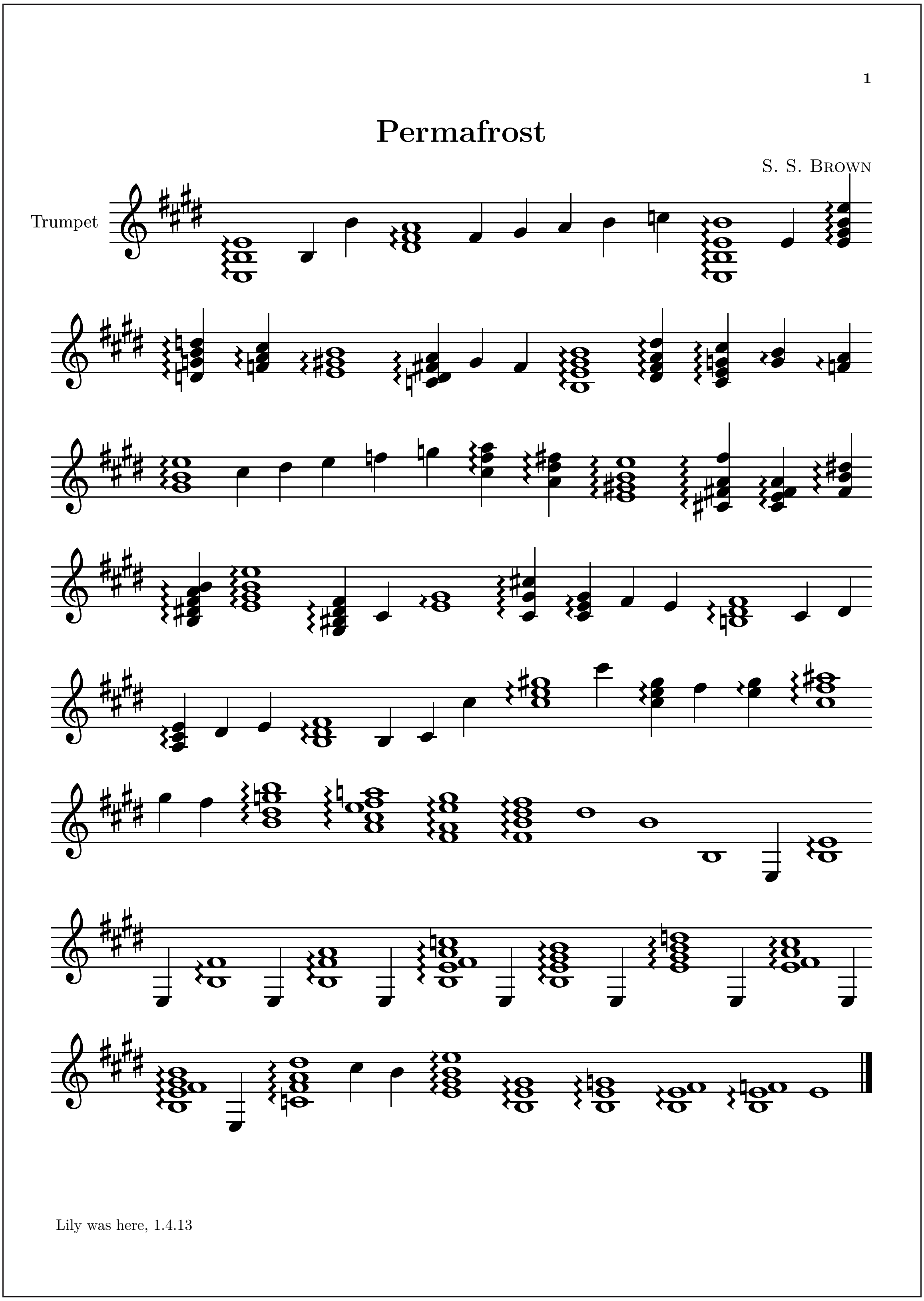
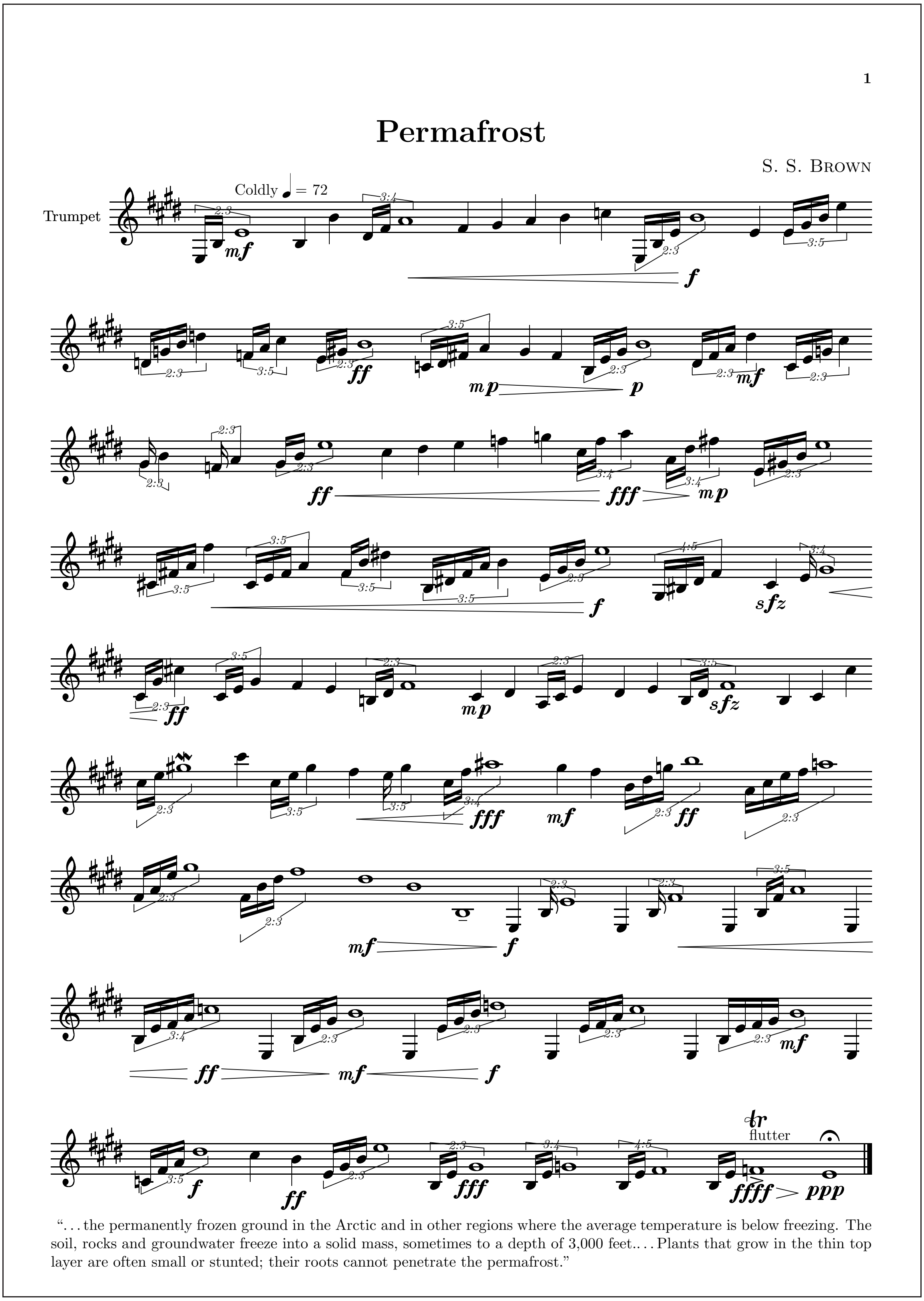


Figure 5.17: A composition in outline form



“. . . the permanently frozen ground in the Arctic and in other regions where the average temperature is below freezing. The soil, rocks and groundwater freeze into a solid mass, sometimes to a depth of 3,000 feet.. . . Plants that grow in the thin top layer are often small or stunted; their roots cannot penetrate the permafrost.”

Figure 5.18: The same composition with more markup added

Transformation 7

Application: Helping a Japanese Koto player to play Irish music, and allowing a Western print-disabled person to produce Japanese Koto notation

Demonstrates: Direct translation to visual positioning instructions; input language invented for the task; different types of output from the same input

Original notation:	Sound recording of Irish music
Source of input:	Text editor
Input language:	An invented music language designed for quick input of that musical style
Output language:	The language of the typesetter Lout
Typeset with:	Lout, with figures generated by CJK-LaTeX
Resulting notation:	Japanese Koto tablature

E natural minor:		E	F♯	G	A	B	C	D	E	F♯	G	A	B	C	D	E
Nogijoshi:	D	E		G	A	B	(♯)	D	E		G	A	B	(♯)	D	E
Koto string no.:	2	3		4	1&5	6		7	8		9	10	11		12	13

E *natural minor* means the Aeolian mode transposed to E. A koto in *Nogijoshi* tuning can play most of the resulting scale and hence can imitate the Aeolian tonality. The koto can sharpen notes but cannot flatten them; therefore it is possible to achieve C as B♯, but it is difficult to get F♯. In the piece in question, both of these notes are relatively unimportant.

Figure 5.19: Using a Japanese scale to produce an ancient Western tonality

5.5.2 Typesetting Japanese koto notation

Preamble. The *koto* is a Japanese musical instrument that is distantly related to the Irish harp. It was desired to arrange some traditional Irish music to be played by a Japanese koto player of elementary standard. This was possible because the Irish melody had a simple rhythm and a tonality that can reasonably be approximated by a common Japanese scale (Figure 5.19).

Koto notation. This is a *tablature* system—it directly represents the strings that should be plucked. Each string is given a number (in Japanese numerals) and these are written in sequence. Timing is usually shown graphically, advancing on the *vertical* axis from top to bottom; horizontal lines are used as a visual guide to show time divisions. Numbers that are adjacent are played at the same time. Other symbols indicate playing techniques. Several columns of this notation can appear on the same page; the rightmost column is read first. The tuning of the strings must be given separately, usually by means of written instructions or a tuning diagram that shows graphically how to arrange the “bridges” on the koto.

Rationale for using conversion. It is difficult for Western composers to write directly in Koto notation, not only because of the Japanese characters and writing direction, but also because there is a conceptual conflict between the Koto numbering system and the numbering systems used in Western music analysis (where 1 is the tonic, 5 is the dominant and so on). A musician who is used to thinking in this numbering could be confused when trying to work with the Koto string numbering, unless the koto player is asked to use an unfamiliar tuning system.

Invented notation. Figure 5.21 shows an invented notation designed for quick input of the Irish music. It would also be possible to use the more complex notation of Figure 5.13, in which case 4DML would have to report as errors (Section 4.7) the use of anything that cannot be represented in koto notation. However, 4DML allows input notations to be customised for each case, and this makes for faster input.

4DML was used to convert this invented notation into input for the document-formatting system Lout [41], which can generate the output from a hierarchy of nested boxes with rotation and scaling—the embedding of 4DML in Lout is shown in Figure 5.20 with comments. There is an implicit transposition between the `hand` and `bar` dimensions. The column breaks were achieved by using standard left-to-right line breaking with rotated boxes on rotated paper (if the paper is rotated anticlockwise, the writing that was in downward columns from right to left becomes writing in rightward rows from top to bottom, which is the standard Western writing direction). This is a common way of achieving vertical text in Western typesetting packages. The resulting Lout code is shown in Figure 5.22 and typeset in Figure 5.23.

4DML was also used to convert the invented notation into input for a Western music program, for viewing and playing as a quick means of checking for mistakes, although an example as small as Figure 5.21 should not need this. The model is partially shown in Figure 5.25 and the result in Figure 5.26.

5.6 Organisation of personal notes

Preamble. Busy people often write personal notes about their plans, contacts and other information in order to take this load off their memories. Writers and designers are also likely to have notes about their ideas. Many who are proficient with computers choose to enter their notes into a computer because of the ease with which they can be stored, copied, changed and reorganised. People with disabilities that impede their handwriting may also choose to make notes electronically.

Several commercially-available computers are small enough to fit in one’s pocket, hence allowing the taking and reviewing of notes almost anywhere (Figure 5.29); most of these can be linked to a desktop computer allowing information to be transferred when necessary. These mobile devices can be used by people with moderate print disabilities provided that the input method is suitable, the software can at least partially be adapted for large print, and the device can be given adequate physical protection if accidents are expected.

```
@Include{koto.setup} @Doc @Text @Begin
@Display clines @Break { @Heading 18p @Font {The Foggy Dew}
(Irish) } (or could read titles from the MML)
Tuning: Nogijoshi @PP (Now follows a preparatory translation table)
[[cml chord export-code / string begin="@OneCol {" end="}" / ( (for each string,)
  note value=D/"@IncludeGraphic kanji2.ps",
  note value=E/"@IncludeGraphic kanji3.ps",
  note value=F/"@IncludeGraphic kanji3.ps", (re-writing F as E)
. . .  (more notes follow)
)]] (End of translation table; start of layout proper)
@RightDisplay -90d @Rotate 21c @Wide {ragged 1.5vx}@Break { (rotate right)
[[cml bar group-size=20 group="} @RightDisplay -90d @Rotate 20c @Wide {ragged
1.5vx}@Break {" /( (for each bar, with new page every 20 bars)
  ]]# Bar [[cml bar count]] (comment the bar number—useful in debugging)
3.9c @Wide @Box { (each bar is a 3.9cm-wide box before rotating right)
      [[cml hand between="//0.1c @FullWidthRule //0.1c"/(]] (for each hand)
          90d @Rotate 2c @Wide { (each hand is a 2cm box rotated left)
              [[cml beat between="//0.1c 1.3c @Wide @LocalWidthRule //0.1c"/(]]
                  @Centre { (two cases—1 or 2 notes in the beat
                      —handle differently because it affects scaling)
                      [[cml note total=1 no-strip call=chord]] (call the
                      [[cml note total=2 no-strip call=chord    translation table)
                              begin="{0.8 0.5} @Scale " between=" // "]]
                  }
              [[cml )]] (end of code for each beat)
                  //0.1c (this Lout code means 0.1cm vertical gap)
          }
      [[cml )]] (end of code for each hand)
  }
[[cml )]] (end of code for each bar)
} @End @Text
```

The literal text is shown in black, the code in blue, and the comments are added here for explanatory purposes only and are not part of CML.

Figure 5.20: 4DML embedded in Lout

```
!block hand
have whitespace , character . as bar beat note string

r,r,r,Bd e,dB,e,dB A,B,D,EF GB,AG,E,ED E,r,r,Bd e,dB,e,dB
A,B,D,EE GB,AG,E,ED E,r,r,EF G,GB,d,cB A,A,B,GA B,g,ed,Bd
e,r,r,Bd e,dB,e,dB A,B,D,EF GB,AG,E,ED E,r,r,Bd

e,dB,e,dB A,B,D,EF GB,AG,E,ED E,r,r,Bd e,dB,e,dB
A,B,D,EE GB,AG,E,ED E,r,r,EF G,GB,d,cB A,A,B,GA B,g,ed,Bd
e,r,r,Bd e,dB,e,dB A,B,D,EF GB,AG,E,ED E,r,r,r
!endblock

!block hand
have whitespace , / character as bar beat note string

r,r,r,r EG,r,EA,r D,r,r,r E,r,r,r r,r,E,r EG,r,EA,r
D,r,r,r E,r,r,r r,r,r,r E,r,EGB,r DE,r,EG,r EG,r,EG,r
EG,r,EG,r EG,r,EG,r E,r,r,r E,r,r,r B/G,EB/G,EGB,r

EG,r,EA,r ED,E/G,E,r E,r,r,r r,E/D,E,r EG,r,EA,r
ED,E/G,E,r E,r,r,r r,r,E,r E/E,E,EGB/E,E DE,E,EG,r
EG,r,EG,r EG,E/E,EG,r EG,r,EG,r E,r,E,r E,r,r,r
GB,E/E,E,r
!endblock
```

Figure 5.21: A Western-style music language for converting to Koto

```
@Include{koto.setup} @Doc @Text @Begin
@Display clines @Break { @Heading 18p @Font {The Foggy Dew}
(Irish) }
Tuning: Nogijoshi @PP
@RightDisplay −90d @Rotate 21c @Wide { ragged 1.5vx } @Break {
# Bar 1
3.9c @Wide @Box {
   90d @Rotate 2c @Wide {
     @Centre {
      @OneCol { @IncludeGraphic kanji0.ps }
     }
    //0.1c 1.3c @Wide @LocalWidthRule //0.1c
     @Centre {
      @OneCol { @IncludeGraphic kanji0.ps }
     }
    //0.1c 1.3c @Wide @LocalWidthRule //0.1c
     @Centre {
      @OneCol { @IncludeGraphic kanji0.ps }
     }
    //0.1c 1.3c @Wide @LocalWidthRule //0.1c
     @Centre {
      {0.8 0.5} @Scale @OneCol { @IncludeGraphic kanji6.ps } //
      {0.8 0.5} @Scale @OneCol { @IncludeGraphic kanji7.ps }
     }
     //0.1c
   }
  //0.1c @FullWidthRule //0.1c
   90d @Rotate 2c @Wide {
     @Centre {
      @OneCol { @IncludeGraphic kanji0.ps }
     }
    //0.1c 1.3c @Wide @LocalWidthRule //0.1c
     @Centre {
      @OneCol { @IncludeGraphic kanji0.ps }
     }
    //0.1c 1.3c @Wide @LocalWidthRule //0.1c
     @Centre {
      @OneCol { @IncludeGraphic kanji0.ps }
     }
    //0.1c 1.3c @Wide @LocalWidthRule //0.1c
     @Centre {
      @OneCol { @IncludeGraphic kanji0.ps }
     }
     //0.1c
   }
 }
# Bar 2
3.9c @Wide @Box {
   90d @Rotate 2c @Wide {
     @Centre {
      @OneCol { @IncludeGraphic kanji8.ps }
     }
    //0.1c 1.3c @Wide @LocalWidthRule //0.1c
     @Centre {
      {0.8 0.5} @Scale @OneCol { @IncludeGraphic kanji7.ps } //
      {0.8 0.5} @Scale @OneCol { @IncludeGraphic kanji6.ps }
     }
    //0.1c 1.3c @Wide @LocalWidthRule //0.1c
     @Centre {
      @OneCol { @IncludeGraphic kanji8.ps }
     }
    //0.1c 1.3c @Wide @LocalWidthRule //0.1c
     @Centre {
      {0.8 0.5} @Scale @OneCol { @IncludeGraphic kanji7.ps } //
      {0.8 0.5} @Scale @OneCol { @IncludeGraphic kanji6.ps }
     }
     //0.1c
   }
  //0.1c @FullWidthRule //0.1c
   90d @Rotate 2c @Wide {
```

Figure 5.22: Automatically-generated Lout code for the Koto tablature (Figure 5.23)

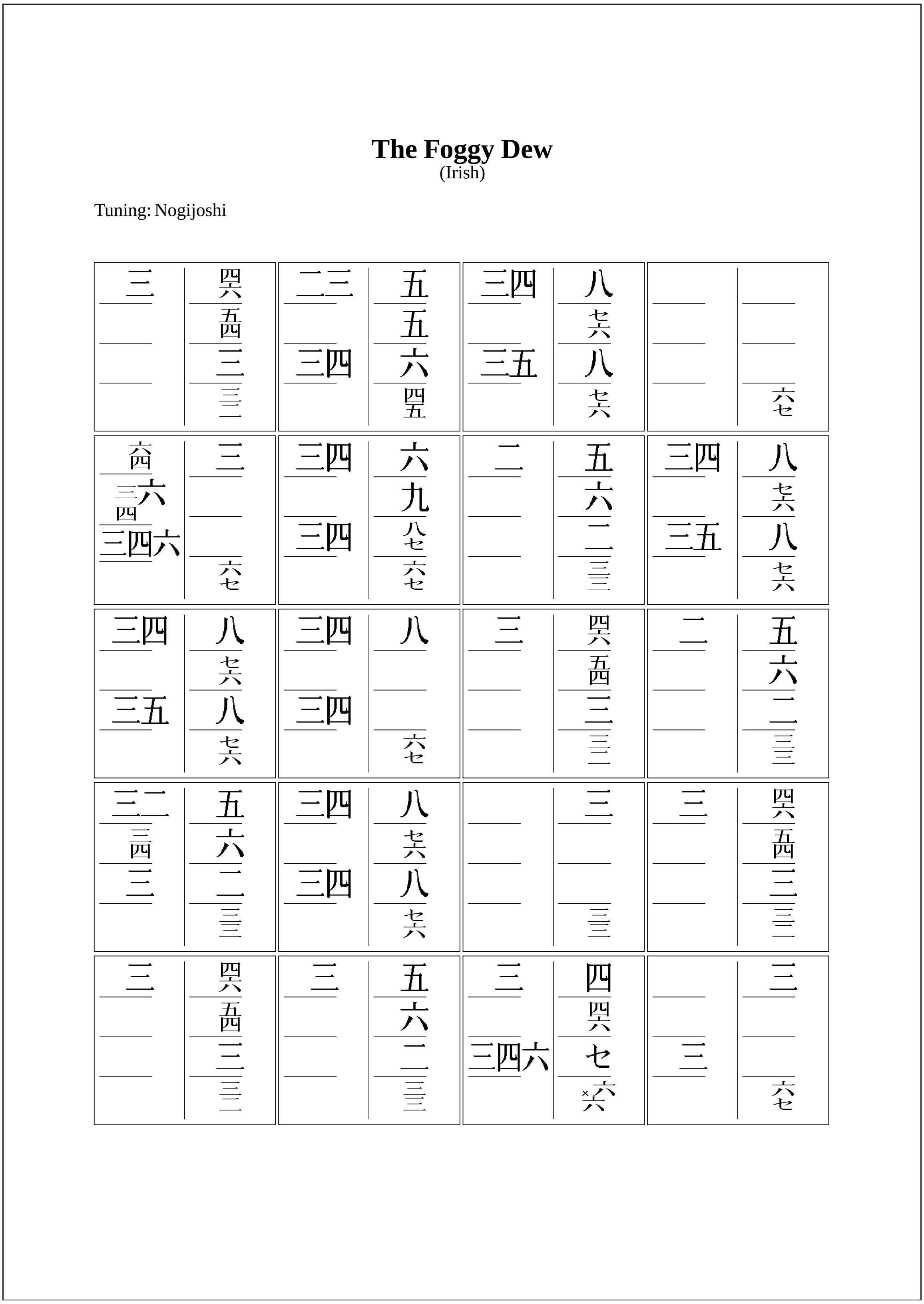


Figure 5.23: Irish music in Japanese Koto tablature

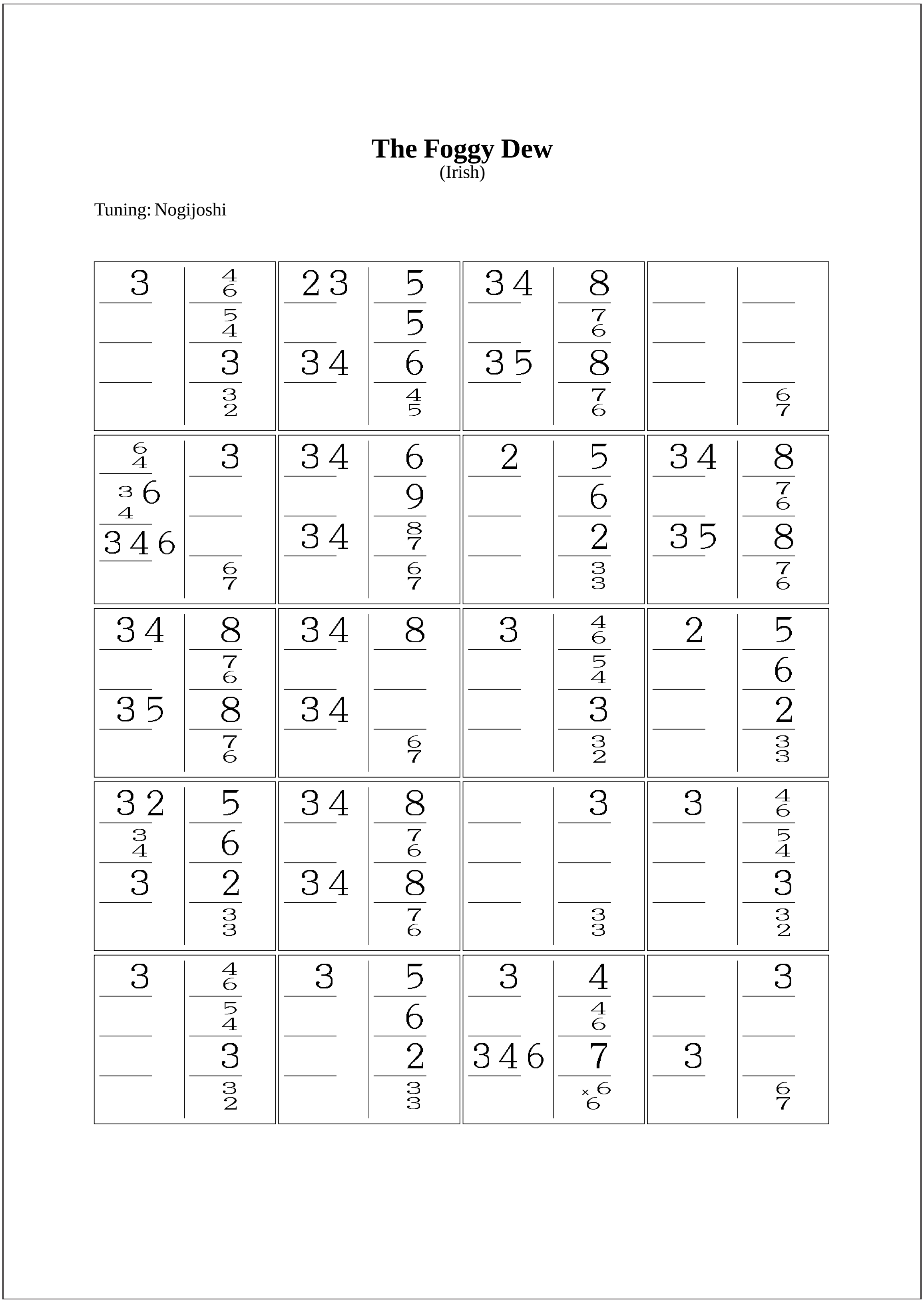


Figure 5.24: As Figure 5.23 but with Arabic numerals (refer to Figure 5.19)

```
J4,4K1#    ; set key and time signature
z3y8#      ; set staves/page

[[cml
  bar between=" "/                                  (space between each bar)
    hand between=","/                               (comma between each hand)
      beat / (                                      (for each beat)
        note total=1 before="L4"/                   (if 1 note in beat, length=4)
             string between="&amp;" no-strip/(     (put ’&’ between notes in chords)
               string value="D" / "o2d",
               string value="E" / "o2e",
               string value="F" / "o2e",
               string value="G" / "o2g",
               ...
             ),
        note total=2 before="L8" call=note          (same for 2 notes but length=8)
      )
]]
```

Figure 5.25: Model to convert Figure 5.21 to a typesetting language for Western musical notation

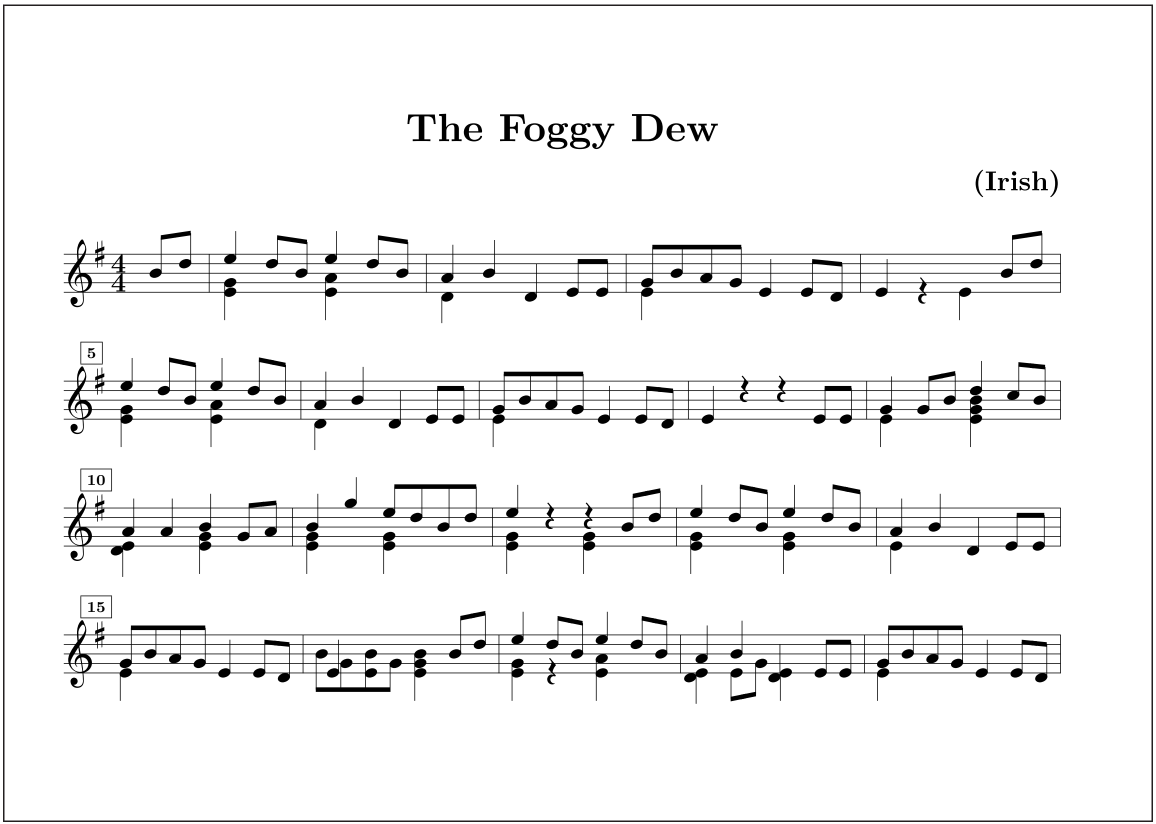


Figure 5.26: The same piece as in Figure 5.23, but in Western musical notation

Transformation 8

Application: Assisting with the re-organisation of large amounts of notes on a restrictive display

Demonstrates: 4DML in an embedded environment

Original notation:	Personal notes in textual form
Source of input:	Text editor on PDA
Input language:	Text with brief markup to direct organisation
Output language:	Plain text
Typeset with:	Any text, speech or Braille terminal
Resulting notation:	Re-organised version of the notes

There are also some mobile devices available for totally blind people, which use speech synthesis or Braille displays, and keyboards that emulate Braille machines (Figure 5.30).

The problem. Reviewing large amounts of notes can be an unwieldy process if they are not well-organised. However, note-taking is often done in haste, which does not facilitate organisation, and the later re-organisation of notes takes time and effort even on a computer. Both the reviewing and the re-organising of notes is made more difficult if large print (or speech or Braille) must be used, because much less information can be displayed at any one time. Conventional time-management software does not address that problem; moving notes around can be a lengthy task even with the aid of keyboard macros.

Categorisation by labelling. A common method of organising notes is to group them into a small number of categories, each of which may have further divisions into categories; usually the broad grouping is done first and the finer grouping later. Conventionally this is done by moving or copying the text from one place to another, which can be time-consuming as explained above. Sometimes it is easier to leave the text where it is and to add *labels* to it (as brief 4DML markup); once the labelling at each level has been completed, 4DML can be used to automatically regroup the text according to the labels.

The 4DML prototype was ported to run on a mobile device and this method was effectively used to organise a large amount of personal notes in large print on that device, with the text editor displaying 4 lines of 20–25 characters at a time. The notes and model were stored in the same file, which resembled Figure 5.27, the resulting organisation being Figure 5.28.

Alternative types of output. Sometimes notes are categorised on several dimensions such as project, priority, timescale, location, or needed resources; in this case the model can be adjusted to group by any of these categories. It is also possible to add HTML or LaTeX markup at the head of each group, so that the result can be used in a Web browser

```
model: project/(project count before="\n\n—", item between="\n")

!block have newline as item
special: × switches project

×thesis Some notes for the thesis
×other Some notes for something else
×thesis More thesis notes
. . .
```

Figure 5.27: Using 4DML to organise personal notes

```
—1
Some notes for the thesis
More thesis notes

—2
Some notes for something else
```

Figure 5.28: The result of transforming Figure 5.27

or hard-copy as well as a text editor—some personal conference notes were effectively reviewed by transforming them into HTML with a generated table of contents and links to facilitate navigation between the sections, and into LaTeX for hardcopy. The HTML links were generated using 4DML's counting facilities (page 64).

5.7 Presentation of diagrams

4DML was used to transform an XML database of some programming languages and their historical relationships (Figure 5.32) into input suitable for AT&T *GraphViz* [26] to draw a directed graph (Figure 5.35). The model is shown in Figure 5.33 and the *GraphViz* code in Figure 5.34.

A separate model (Figure 5.36) was used to derive a subset of the database suitable for producing a graph that could fit in a limited space without reducing the print size (Figure 5.37). The model specifies "ML" as a starting point and instructs 4DML to find languages that had influenced it; this is done using the technique from Section 4.6.2 (see Figure 4.28 on page 65), with automatically-generated `link` elements that connect any duplicate strings (the text `link` is specified using a 4DML command-line switch).

A modified version of this model (Figure 5.38) produced text suitable for a speech synthesizer (Figure 5.31).

Figure 5.29: Mobile device that can be used for taking notes (picture courtesy of Psion PLC)

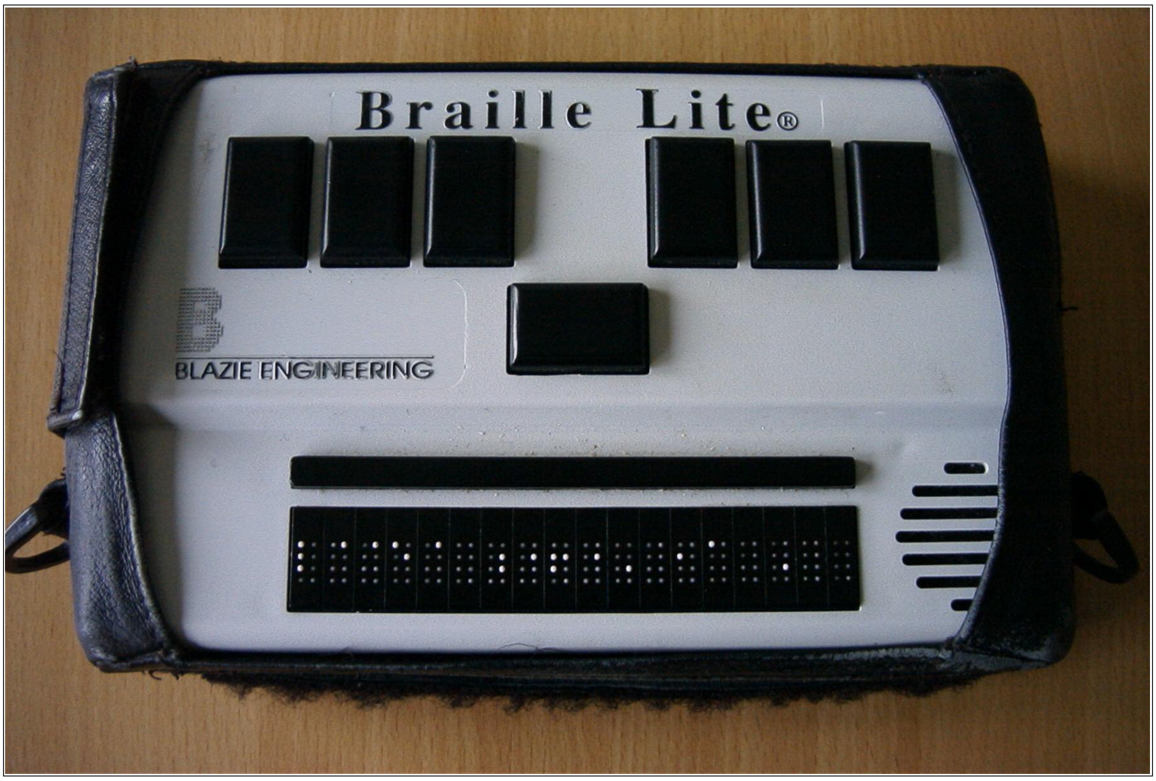


Figure 5.30: Braille-based note-taking device (photo: Hans Schou)

Transformation 9

Application: Creating relationship diagrams or subsets of them, for presentation in normal or large print

Demonstrates: Link traversal

Original notation:	Database of programming languages and their historical relationships
Source of input:	Text editor
Input language:	XML
Output language:	The language of AT&T's GraphViz
Typeset with:	AT&T GraphViz (aka Dot)
Resulting notation:	Complete or partial relationship diagram

Transformation 10

Application: Presenting a subset of a diagram to users of speech synthesis

Demonstrates: Different types of output from the same input

Original notation:	Database of programming languages and their historical relationships
Source of input:	Text editor
Input language:	XML
Output language:	Plain text
Typeset with:	Any text, speech or Braille terminal
Resulting notation:	English description of relationship diagram

Speech synthesizer: "ML (1983) draws from LISP. LISP (1958) draws from FLPL. FLPL (1958) draws from IPL 2 and FORTRAN. IPL 2 (1956). FORTRAN (1954). "

Figure 5.31: As Figure 5.37 but formatted for speech synthesis

5.8 Other applications of 4DML

Although 4DML is primarily intended for the transformation of notations, it does have other potential applications, as will be briefly demonstrated here.

5.8.1 Generalised error checking

Takata et al [65] presented a technique for automatically checking websites for potential accessibility problems. They had actually designed a generalised system, which can check *any* XML document against *any* set of guidelines. The system works by converting the guidelines (Figure 5.39) into an XSLT stylesheet, which then transforms the XML document into an error report (Figure 5.40). This is a novel use of a generalised transformation tool—the error report is the output notation.

Since the XSLT code is automatically generated from a higher-level language, the usability of XSLT itself is not necessarily an issue. However, the language that was used to specify the guidelines relied heavily on XPATH expressions, which in some cases have limited expressiveness. For example, the problem of checking that HTML links were consistently described (i.e. two links to the same page have the same description) required XPATH expressions that were so complex that Takata et al decided to extend the XPATH language for this specific problem.

As a proof of concept, the system was partially reproduced using 4DML instead of XSLT, by using the error-checking techniques described in Section 4.7. The result (Figure 5.41) was able to produce reports that indicated the context of the error. 4DML could also utilise link traversal (Section 4.6.2) to check the consistency of repeated information.

```
<?xml version="1.0"?>
<document>
<language id="ada">
<name>Ada</name>
<year>1978</year>
<drawsfrom id="pascal" />
<drawsfrom id="algol68" />
<drawsfrom id="pl1" />
</language>
<language id="algol60">
<name>Algol 60</name>
<year>1960</year>
</language>
<language id="algol68">
<name>Algol 68</name>
<year>1968</year>
</language>
<language id="ansic">
<name>ANSI C</name>
<year>1989</year>
</language>
<language id="b69">
<name>B</name>
<year>1969</year>
</language>
<language id="bcpl">
<name>BCPL</name>
<year>1967</year>
</language>
<language id="c99">
<name>C99</name>
<year>1999</year>
</language>
<language id="c">
<name>C</name>
<year>1971</year>
</language>
<language id="commonlisp">
<name>Common Lisp</name>
<year>1984</year>
<drawsfrom id="maclisp" />
<drawsfrom id="interlisp" />
</language>
<language id="cpparm">
<name>C++ (ARM)</name>
<year>1990</year>
<drawsfrom id="ml" />
<drawsfrom id="ada" />
<drawsfrom id="clu" />
<drawsfrom id="ansic" />
</language>
<language id="cpp">
<name>C++</name>
<year>1983</year>
<drawsfrom id="simula67" />
<drawsfrom id="algol68" />
</language>
<language id="cwithclasses">
<name>C with Classes</name>
<year>1980</year>
</language>
<language id="flpl">
<name>FLPL</name>
<drawsfrom id="ipl2" />
<drawsfrom id="fortran" />
<year>1958</year>
</language>
<language id="fortran66">
<name>FORTRAN 66</name>
<year>1966</year>
</language>
<language id="fortran77">
<name>FORTRAN 77</name>
<year>1977</year>
</language>
<language id="fortran90">
<name>FORTRAN 90</name>
<year>1990</year>
</language>
<language id="fortran95">
<name>FORTRAN 95</name>
<year>1995</year>
</language>
<language id="fortraniii">
<name>FORTRAN III</name>
<year>1958</year>
</language>
<language id="fortranii">
<name>FORTRAN II</name>
<year>1958</year>
</language>
<language id="fortraniv">
<name>FORTRAN IV</name>
<year>1961</year>
</language>
<language id="fortran">
<name>FORTRAN</name>
<year>1954</year>
</language>
<language id="interlisp">
<name>Interlisp</name>
<year>1978</year>
</language>
<language id="ipl2">
<name>IPL 2</name>
<year>1956</year>
</language>
<language id="isocpp">
<name>ISO C++</name>
<year>1998</year>
```

Figure 5.32: XML data—historical relationships between some programming languages (supplied by Rob Hague)

```
digraph PLFT {

[[cml language no-strip/(]]
  [[cml id]] [label="[[cml name]]\n([[cml year]])"];
  [[cml drawsfrom/(]]
    [[cml id, " -> ", language broaden/id]] [color=gray, weight=0.1];
  [[cml ), descends/(]]
    [[cml id, " -> ", language broaden/id]] [style=bold];
[[cml ) ) ]]

}
```

Figure 5.33: Model to convert Figure 5.32 to GraphViz code

5.8.2 Statistical reporting

4DML has been used to assist with the statistics work in an applied linguistics research project. The linguistics researcher had collected over 1000 questionnaire responses and categorised them on 5 dimensions (questionnaire number, question number, general type of response, nationality and gender), all of which were potentially significant in the analysis. Where there were multiple responses, or where a response matched more than one category, this information was also coded.

Originally the coding was done in a very large spreadsheet, but this became unwieldy; many hours were spent navigating around the spreadsheet and developing the necessary formulae to count the results according to various criteria. The research student conducting the experiment was unsure about how the data should be analysed, so the requirements for these formulae constantly changed. Eventually, the numerous "copy and paste" operations took their toll—an undetected error in cursor positioning caused the data to become corrupted, resulting in much frustration.

The data was then encoded into 4DML's Matrix Markup Language, which facilitates multi-dimensional encoding as has previously been shown. 4DML was then used as a simple database query language to select appropriate items for counting. This allowed different analyses to be experimented with much more rapidly than the spreadsheet approach. Reports were produced by embedding 4DML's Compact Model Language in an HTML file. Since 4DML does not provide mathematical functions, these were implemented by having 4DML output JavaScript (a common embedded scripting language for Web browsers) which was then evaluated to display the final result.

It would be possible to extend 4DML with the additional functionality needed so that post-processing is not necessary; however, since 4DML is primarily a tool for the conversion of notations, such extension is arguably not necessary as it can be provided by coupling 4DML with other tools as described above.

The linguistics researcher has asked to remain anonymous for personal reasons, and has requested that further details are not shown in this thesis.

5.9 Summary

This chapter illustrated example 4DML transformations from a variety of applications, including online data, mathematics, music, time management and diagrams, plus special-

```
digraph PLFT {

ada [label="Ada\n(1978)"];
  pascal -> ada [color=gray, weight=0.1];
  algol68 -> ada [color=gray, weight=0.1];
  pl1 -> ada [color=gray, weight=0.1];
  algol60 [label="Algol 60\n(1960)"];
  algol68 [label="Algol 68\n(1968)"];
  algol60 -> algol68 [style=bold];
ansic [label="ANSI C\n(1989)"];
  krc -> ansic [style=bold];
b69 [label="B\n(1969)"];
  bcpl -> b69 [style=bold];
bcpl [label="BCPL\n(1967)"];
  cpl -> bcpl [style=bold];
c99 [label="C99\n(1999)"];
  ansic -> c99 [style=bold];
c [label="C\n(1971)"];
  b69 -> c [style=bold];
commonlisp [label="Common Lisp\n(1984)"];
  maclisp -> commonlisp [color=gray, weight=0.1];
  interlisp -> commonlisp [color=gray, weight=0.1];
  cpparm [label="C++ (ARM)\n(1990)"];
  ml -> cpparm [color=gray, weight=0.1];
  ada -> cpparm [color=gray, weight=0.1];
  clu -> cpparm [color=gray, weight=0.1];
  ansic -> cpparm [color=gray, weight=0.1];
  cpp -> cpparm [style=bold];
cpp [label="C++\n(1983)"];
  simula67 -> cpp [color=gray, weight=0.1];
  algol68 -> cpp [color=gray, weight=0.1];
  cwithclasses -> cpp [style=bold];
cwithclasses [label="C with Classes\n(1980)"];
  krc -> cwithclasses [style=bold];
simula67 -> cwithclasses [style=bold];
flpl [label="FLPL\n(1958)"];
  ipl2 -> flpl [color=gray, weight=0.1];
  fortran -> flpl [color=gray, weight=0.1];
  fortran66 [label="FORTRAN 66\n(1966)"];
  fortraniv -> fortran66 [style=bold];
fortran77 [label="FORTRAN 77\n(1977)"];
  fortran66 -> fortran77 [style=bold];
fortran90 [label="FORTRAN 90\n(1990)"];
  fortran77 -> fortran90 [style=bold];
fortran95 [label="FORTRAN 95\n(1995)"];
  fortran90 -> fortran95 [style=bold];
fortraniii [label="FORTRAN III\n(1958)"];
  fortranii -> fortraniii [style=bold];
fortranii [label="FORTRAN II\n(1958)"];
  fortran -> fortranii [style=bold];
fortraniv [label="FORTRAN IV\n(1961)"];
  fortraniii -> fortraniv [style=bold];
```

Figure 5.34: GraphViz code from Figure 5.33

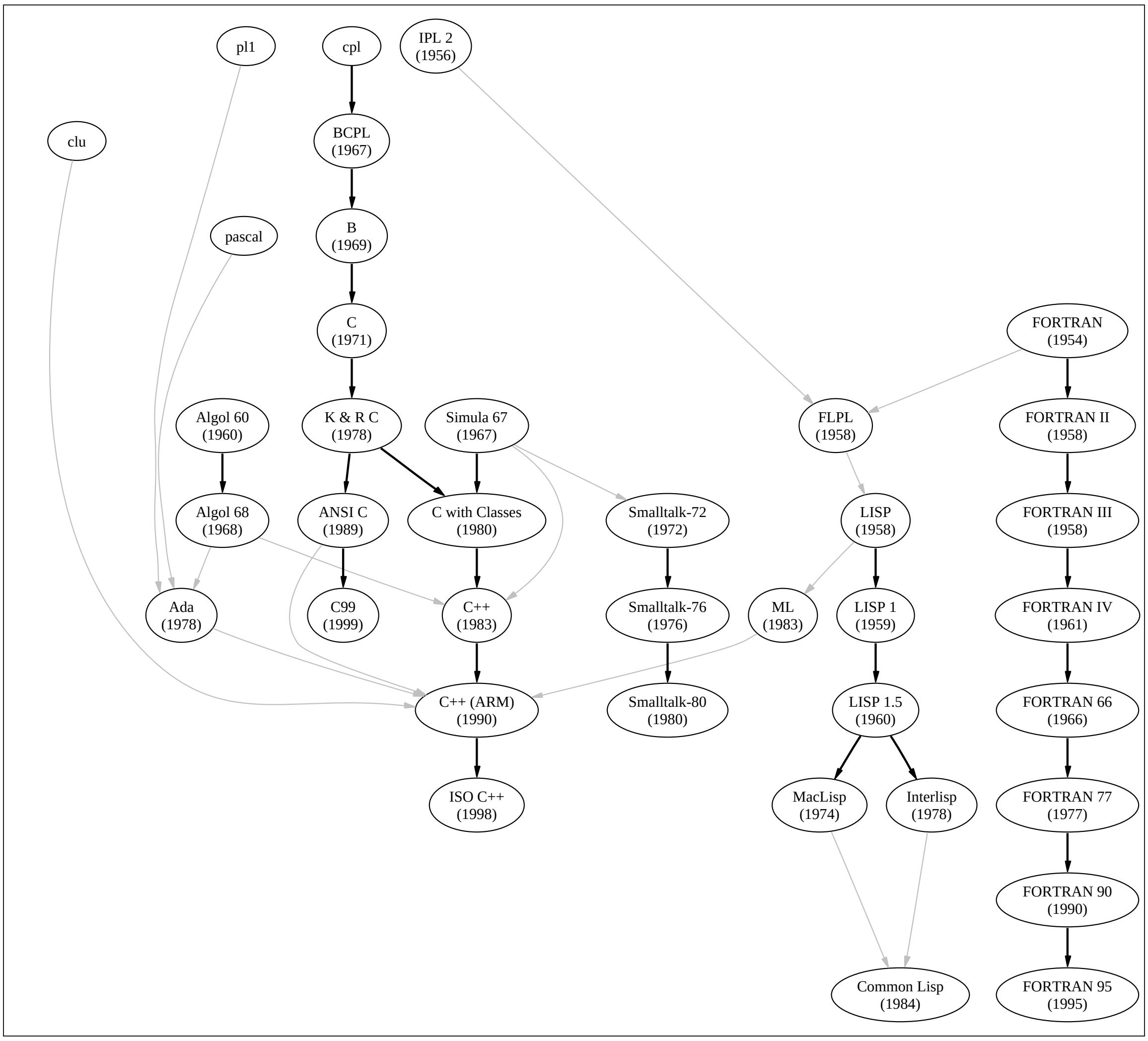


Figure 5.35: Result of inputting Figure 5.34 into GraphViz

```
digraph PLFT { rankdir="LR"

[[cml library export-code/
   doANode/(]]
  [[cml id]] [label="[[cml name]]\n([[cml year]])"];
  [[cml drawsfrom external=never/(]]
    [[cml id, " -> ", language broaden/id]] [color=gray, weight=0.1];
    [[cml id/link broaden/id/language broaden call=doANode ) ),
language no-strip/id value=ml/language broaden call=doANode
]]
}
```

Figure 5.36: Model to convert a *subset* of Figure 5.32 to GraphViz code

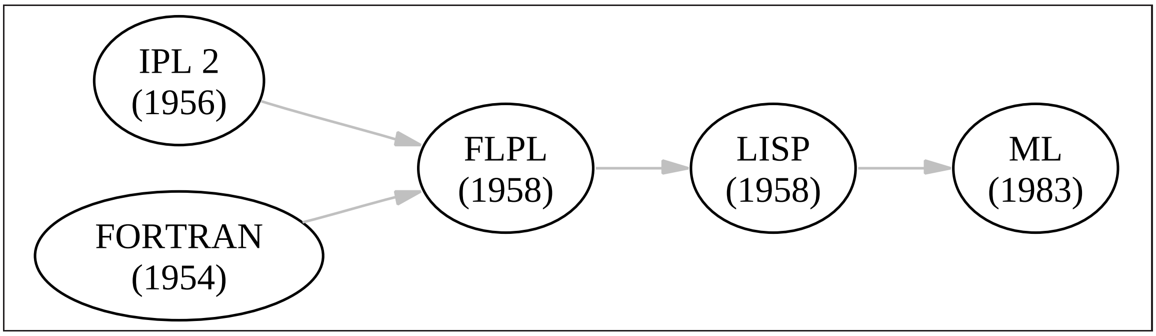


Figure 5.37: Subset of Figure 5.35, suitable for large print

```
[[cml library export-code/
   doANode/(
  name, " (", year, ")",
  drawsfrom external=never before=" draws from " between=" and "/
    (id/link broaden/id/language broaden/name),
  ". ",
  drawsfrom external=never/
    (id/link broaden/id/language broaden call=doANode)
  ),
language no-strip/id value=ml/language broaden call=doANode
]]
```

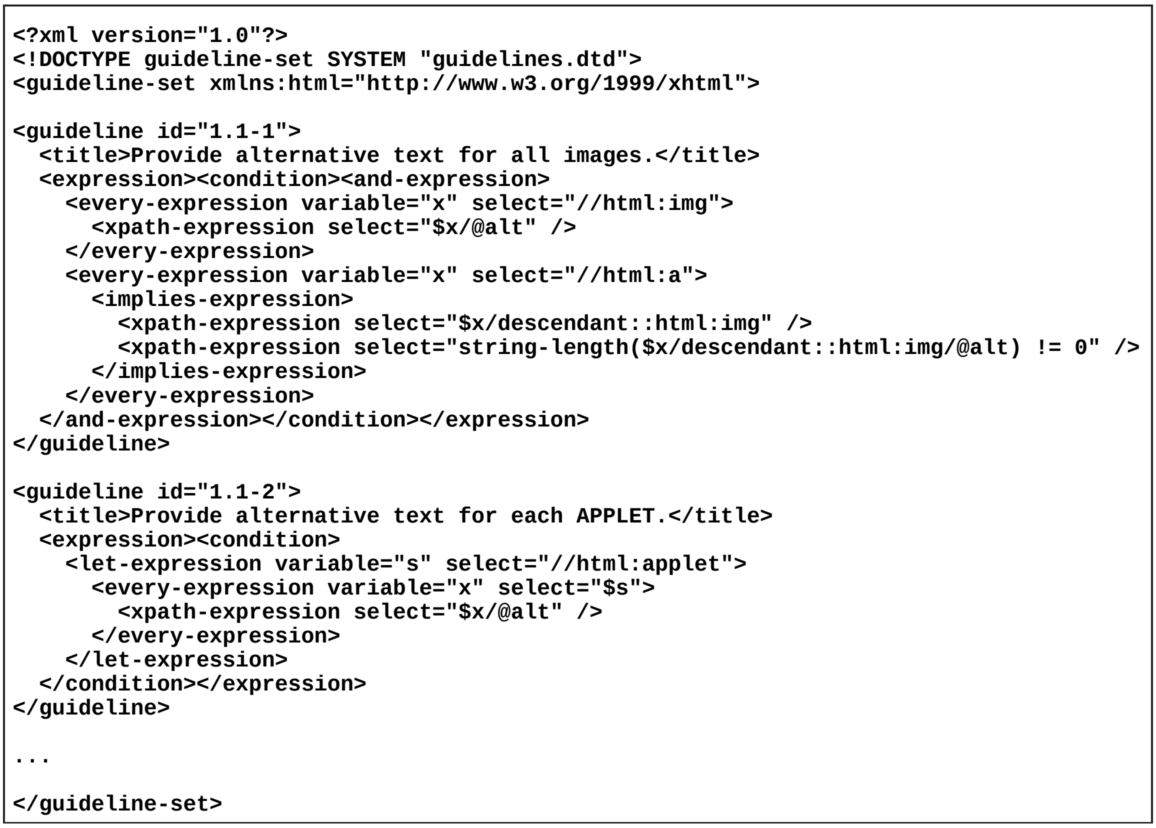
Figure 5.38: Model to convert a subset of Figure 5.32 to a textual description

needs typesetting for language learning. It also introduced two potential applications not directly related to notation conversion—generalised error checking and database reporting.

In each case, it is evident that different types of output can be derived from a common source, using CML code that is easily written. It is also evident that different forms of input can be used to accomplish the same results. Both input and output are easily customisable to users' needs.

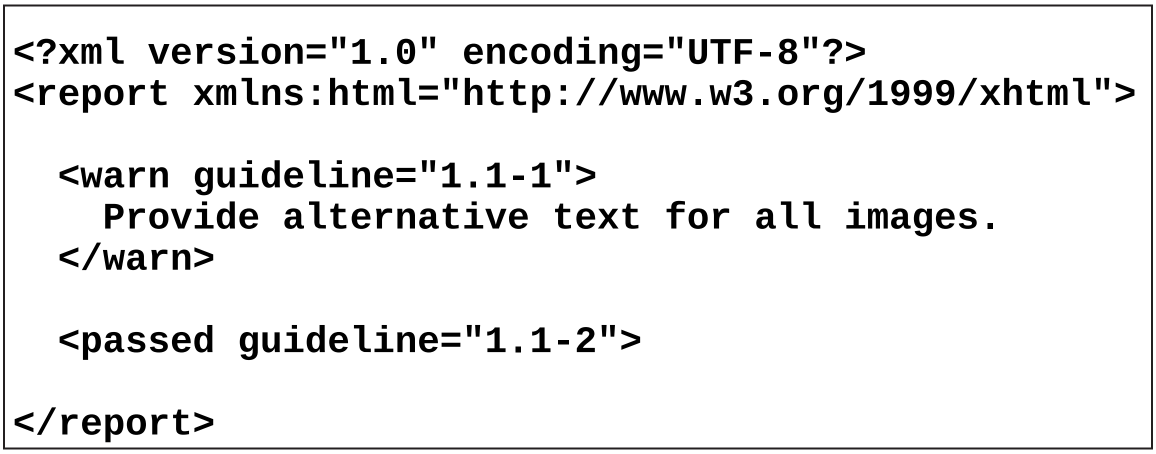
Perhaps one of the more novel approaches exhibited is that of "distributed" or "aspect-oriented" music encodings, in imitation of aspect-oriented programming. Different aspects of the music, such as note letters, octaves, durations, enharmonics, ornaments, etc, are coded on separate passes through the score, and the model interleaves them when producing output. This facilitated the transcription of already-written music because the user need consider only one aspect at a time, avoiding the need to switch rapidly between many different features of a complex input language. Aspect-oriented encoding also proved beneficial for original composition, the different aspects of the composition being added at different times and converted into musical notation by the user's model. The method likewise holds potential for music publishers and repositories, because it could be used to divide encoding skills among several people.

An evaluation of 4DML is presented in the next chapter.



```
<?xml version="1.0"?>
<!DOCTYPE guideline-set SYSTEM "guidelines.dtd">
<guideline-set xmlns:html="http://www.w3.org/1999/xhtml">

<guideline id="1.1-1">
  <title>Provide alternative text for all images.</title>
  <expression><condition><and-expression>
    <every-expression variable="x" select="//html:img">
      <xpath-expression select="$x/@alt" />
    </every-expression>
    <every-expression variable="x" select="//html:a">
      <implies-expression>
        <xpath-expression select="$x/descendant::html:img" />
        <xpath-expression select="string-length($x/descendant::html:img/@alt) != 0" />
      </implies-expression>
    </every-expression>
  </and-expression></condition></expression>
</guideline>

<guideline id="1.1-2">
  <title>Provide alternative text for each APPLET.</title>
  <expression><condition>
    <let-expression variable="s" select="//html:applet">
      <every-expression variable="x" select="$s">
        <xpath-expression select="$x/@alt" />
      </every-expression>
    </let-expression>
  </condition></expression>
</guideline>

...

</guideline-set>
```

Figure 5.39: Part of the Web content accessibility guidelines in the error-checking language developed by Takata et al [65]



```
<?xml version="1.0" encoding="UTF-8"?>
<report xmlns:html="http://www.w3.org/1999/xhtml">

  <warn guideline="1.1-1">
    Provide alternative text for all images.
  </warn>

  <passed guideline="1.1-2">

</report>
```

Figure 5.40: Website error report from the guidelines in Figure 5.39

```
html wildcard=any-element/(

  img/alt expected,
  a/img/(
    alt value=""/error expected,
    alt other-values expected),

  applet/alt expected,

  ...

  any-element call=html)
```

Figure 5.41: 4DML version of Figure 5.39

6 Evaluation

This chapter discusses an evaluation of the 4DML framework. Since 4DML has a wide range of potential applications, as shown by the illustrative examples in Chapter 5, a method is needed for performing a general evaluation that is not limited to a few particular cases of its use. The Cognitive Dimensions analysis, introduced in Section 6.1, is such a method.

Chapter overview. This chapter introduces Cognitive Dimensions of Notations and uses it to compare 4DML with other generalised transformation frameworks from Chapter 3. It then elaborates on two minor sources of error-proneness in the 4DML framework.

6.1 4DML and Cognitive Dimensions of Notations

Cognitive dimensions of notations [53, 27] is a usability analysis framework that models programming languages, other software, interaction devices and appliances as *information artefacts* which provide environments for viewing, navigating and altering a notation of some kind. The usability of a system is affected by both the notation and the environment. In order to include people with disabilities and other special needs in the analysis, it may be necessary to represent the person's senses, nervous system and so on as part of that environment—for example, if the system is said to provide "visibility" of certain information then this will only be true if the method that it uses is compatible not only with whatever hardware is in use but also with the person's possibly-limited senses; similarly, if the system is said to provide low viscosity (low resistance to change) then the user's ability to manipulate the controls to perform the change must be accounted for. Thus a notation-environment pair that is well-suited to one person for a certain task might not be as well-suited to another person even for the same task.

Transformation systems such as 4DML can be seen in the context of larger systems as redefinition devices, allowing other notations to be changed. If the transformation system itself is sufficiently usable for the person or group of people in question, then it can increase the usability of the larger environment by adjusting the notation so that it is better suited to the tasks at hand, within the context of the environment that must be used. Since the environment is often constrained in ways that are prohibitively difficult to change (e.g. new hardware cannot be acquired, a disability cannot be cured, etc), adjusting the notation might be the only way of adjusting the usability of a system in some circumstances. Although the environment might be changed with some limited degree of freedom, it will often be the case that the only feasible method of improving usability is to optimise the notation for the tasks at hand in the context of the environment that must be used, and this emphasises the importance of transformation systems.

Redefinition devices such as transformation systems can themselves be analysed using the cognitive dimensions framework; in this case the notation and environment in question is that which is used to describe or program the desired transformations. Thus we can use the cognitive dimensions framework as an aid to evaluating the usability of the 4DML system and comparing it with other generalised transformation frameworks (Chapter 3).

As a first approximation, the following assumptions about the environment might be made when disabled people are involved:

1. that specialised, graphical integrated development environments (IDEs) for the notation are likely to be incompatible with the person's limitations and established way of working, and should consequently be regarded as unavailable;

2. that some kind of text editor is available, but the amount of text that is concurrently displayed, and/or the ease of inputting it, might be severely limited due to the constraints of the environment, such as the requirement to use large print, Braille, speech and/or a limited display area.

This leads to the two requirements of textual brevity and readability:

Brevity. The act of reading text takes effort, and this effort is increased if the user has a reading difficulty, or is using speech synthesis, or if the display can only accommodate a small number of characters, which is often the case with large-print and Braille displays as well as mobile computers. The act of entering text also takes effort, particularly if the user has a motor disability or is using a slow input method. It is therefore desirable to make the code as concise as possible.

Readability. While the text needs to be concise, its entropy must not be too high, otherwise it would require much effort to decipher. Ideally it should be possible to read and write code in a linear fashion and at a fairly constant rate, without compromising its understandability. Code that relies on special punctuation characters or layout can be particularly cumbersome for users of voice recognition and speech synthesis, and can also present problems for those who read visually but who rely on the redundancy of spelt-out words. However, flexibility is needed because punctuation and layout *can* aid readability in some cases, particularly when the editor can assist with the layout and highlight the syntax.

Brevity and readability should be possible for all classes of transformation that one might reasonably expect to encounter when converting between notations; it should particularly be possible for cases where it is difficult to accomplish the same in other transformation frameworks.

The requirements of brevity and readability can be expressed in terms of many of the cognitive dimensions, such as diffuseness, closeness of mapping, and consistency. The following table gives a brief analysis of 4DML and other generalised transformation frameworks from Chapter 3, with respect to 14 cognitive dimensions.

<table>
<tr><th>Cognitive dimension</th><th>4DML</th><th>XML transformation tools</th><th>Parsers and rewriting systems</th><th>Unix tools</th></tr>
<tr><td>Premature commitment (constraints on the order of doing things; user is forced to make irreversible decisions before the task is complete)</td><td colspan="4">Very little in an ordinary text editor</td></tr>
<tr><td>Hidden dependencies (important but invisible links between different things)</td><td>Dependencies in nomenclature between input parsing and output model, but relatively little structural dependency due to the dimension-independent way that the internal data is read (re-structuring the input won't usually require the model to be changed and vice versa)</td><td colspan="3">Rules are heavily dependent on the structure of the input; changing the input structure will almost always require the rules to be re-written.</td></tr>
<tr><td>Secondary notation (support for comments and so forth)</td><td>Yes</td><td>Yes</td><td>Yes</td><td>Yes</td></tr>
<tr><td>Viscosity (difficulty of changing the notation)</td><td colspan="4">Very little in an ordinary text editor, but notations that are difficult to work with due to other dimensions can also be regarded as being more viscous</td></tr>
<tr><td>Visability (able to see as much of the notation as is necessary for the task, including different parts at the same time if necessary)</td><td colspan="4">See Diffuseness. In a text editor, notations that are diffuse (verbose) require more display area; if display is severely limited then this will likely impair visability.</td></tr>
</table>

Cognitive dimension	4DML	XML transformation tools	Parsers and rewriting systems	Unix tools
Closeness of mapping (of the notation to what it ultimately represents)	MML input parsing is close to input structure; models closely follow structure of desired output	Transformation code maps to the transformation method rather than to the desired output, although it is not excessively far from it	Input structure is mapped fairly closely by good grammar language; transformation code maps to the transformation method rather than to the desired output; can require intermediate states	Transformation code maps to the transformation method rather than to the desired output; can require many intermediate states and other complexity
Consistency (similar meanings are expressed similarly)	No known major inconsistencies, although a better choice of names for some parameters might be possible	XSLT depends on XPATH (an entirely different, non-XML based language) for specifying selections; can require significant additional learning	Fairly consistent	Many inconsistencies when two or more separate tools are in use, which might employ entirely different languages and modes of operation, although some ideas are common across much of Unix
Diffuseness (verbosity of language)	Tends to be brief	Can be very verbose without a specialised IDE	Relatively verbose	Can be fairly brief if the tools are well-chosen
Error-proneness (notation and/or environment invites mistakes, especially when being used hastily)	Errors such as incorrect choice of parameters are occasionally possible	Incorrect use of punctuation characters in XPATH is possible	Few errors likely from hasty input, but see “hard mental operations” below	Incorrect use of punctuation characters etc is frequently possible

Cognitive dimension	4DML	XML transformation tools	Parsers and re-writing systems	Unix tools
Hard mental operations	May need to write models carefully, especially when starting to learn 4DML, but they reflect the structure of the desired output, so the total amount of mental operation should be less than that for systems where more indirection is needed	Sometimes unclear how to perform certain operations e.g. transposition; XPATH can be difficult	Frequently cope with only a subset of BNF grammars; ensuring that the grammar lies within this subset can be difficult. Composing the re-writing rules can also present problems.	Sometimes have to implement algorithms using them if the desired operations are not readily available
Progressive evaluation (can check partially-completed work)	Usually possible	Often possible	Often difficult for parsers; a near-complete grammar must be constructed before anything will parse. More possible with rewriting rules.	Sometimes possible
Provisionality (ability to sketch things without commitment)	Not usually a problem in an ordinary text editor			
Role-expressiveness (not difficult to infer the purpose and effects of components)	Fairly expressive, although there is some compromise for brevity	XSLT is quite expressive; XPATH not so much	Expressive so long as meaningful identifiers are used	Varies with different tools; sometimes very difficult to read the notation without significant prior knowledge
Abstraction (supports abstraction mechanisms, not unnecessarily forced to use them)	Support for libraries of models and model fragments	Support for subroutines (as template rules that can be called) and for importing libraries of them	Abstraction possible by using non-terminals of the grammar; can be more difficult with the re-writing rules	Most (but not all) tools support some means of abstraction

```
!block special-instructions-for-each-word
have newline , as line word

shout, say softly, speed up, ←this comma is incorrect
look at the ceiling, slow down, gasp
!endblock
```

Figure 6.1: A possible error in MML

6.2 Error-proneness in 4DML

This section discusses two aspects of the 4DML framework that are known potential sources of error—MML separators, and certain model parameters. These points of error-proneness are relatively minor but they are difficult to eliminate completely.

6.2.1 A subtlety in Matrix Markup Language

Matrix Markup Language (Section 4.5) provides a means of inputting matrix-like blocks of data with user-defined separators. For example, a definition

```
have newline , as page item
```

will cause a newline character to separate `pages`, and `,` to separate `items` within each `page`. Meanwhile, the definition

```
have newline as page
also have , as item
```

has the same effect, except that starting a new `page` does *not* start a new `item`. If one wants to signify that both a new `page` and a new `item` are to be started, then one must input both a `,` and a newline. With the first definition, on the other hand, the extra `,` should *not* be input, as it will cause a spurious blank `item`.

Normally this distinction will not cause problems—if the application is such that an `item` is conceptually a subdivision of a `page`, then the first definition will be used and the extra `,` will naturally be omitted. If on the other hand the `item` divisions are completely independent of the `page` divisions, the second definition will be used and the extra `,` will naturally be included. However, there are two potential sources of difficulty:

1. If a new `page` *does* always start a new `item`, but an `item` is not *conceptually* a subdivision of a `page`, then a user might use the first definition but accidentally include an extra `,` (see Figure 6.1).

2. The second definition (new `page` does *not* start new `item`) can be confusing when used with whitespace separators (Figure 6.2).

4DML will detect such errors in its consistency checks (Section 4.7), but it is unlikely that they can be avoided altogether without overcomplicating the semantics of MML.

```
!block bass-notes
have whitespace as chord
also have newline as bar

1 5 6 4 ←need an extra space as well as this newline
#4 2 5-
!endblock
```

Figure 6.2: Another possible error in MML

6.2.2 A potential ambiguity in model terminology

The 4DML model parameters `start-at`, `end-at`, `number` and `count` (page 64+) are potentially misleading. These parameters refer to the numerical positions of elements in the input. By default, the numbers used are the *original* element positions—the first child of a parent element is element number 1, the second child is number 2 and so on.

Consequently, the model:

```
part number=1/...
```

does not mean “do this for part 1”. It means “do this for the `part` element that is in position number 1”. There is a difference: If the element in position number 1 is not a `part` (e.g. it is a `title`—see Figure 4.17 on page 59), then `part number=1` will do nothing, because there is no `part` with position number 1. In Figure 4.17 the first `part` has position number 2.

If what is meant is “do this for part 1”, then the model should be:

```
part renumber number=1/...
```

which will (a) select all `part` elements, (b) renumber them starting from 1 and (c) select number 1 in the new numbering—see Figure 4.29 on page 65.

Why not renumber by default? It would be possible to `renumber` by default and introduce a `keep-position` parameter to override this behaviour. This may seem to make `part number=1` more directly descriptive. However, consider this variation:

```
part no-strip / bar / (note ..., part number=1 / ...)
```

The intent of this is,

```
for each “part” p, select p and
  for each “bar” b, select b and
    Do something with the notes
    if p had position 1
      then Do something special
           (e.g. write out the directives that apply to all parts)
```

However, by the time the innermost `part` (corresponding to the **`if`**) is reached, there will be only one `part` in scope, because the code is being executed for each part *in turn*. If `number` implies `renumber` then *each* of these `part`s will be renumbered to 1, which is not what the model writer intended.

Tradeoff. The above demonstrates that a decision of whether or not to make `renumber` the default behaviour involes a tradeoff—it would make some models easier to write and others more difficult. The second of the above two cases occurs more frequently in practice, so a default behaviour of *not* renumbering leads to fewer misunderstandings than a default behaviour of renumbering, and is less likely to lead to errors that are insidious (difficult to detect).

Other possible approaches include:

1. Invent complex rules that decide the default according to context, perhaps including a renumbering with respect to data outside the current subset. This could lead to errors that, while rare, are even more insidious—it is probably better to keep the process as understandable as possible.

2. Provide *no* default—every time numbering is used, force the user to state explicitly whether or not the elements should be renumbered. This would sacrifice brevity in English, although other human languages may provide words that express such concepts as "the original position number" and "the adjusted position number" more succinctly.

Defence of 4DML's approach. Words in a natural language have many meanings, connotations and cultural implications. These can change over time. The use of complex, technical words may avoid the accidental invocation of misleading mental associations, but this has its drawbacks. Since 4DML is designed for brevity, it should not introduce overly complex terminology merely to avoid ambiguity; it would be better to use short, relatively simple keywords, clarifying their meanings where necessary in prominent parts of the documentation. If a user finds a term particularly awkward then it can be redefined.

This author does not claim to have found "perfect" terminology for the parameters—such would not be possible, since it would depend on the language, culture, generation and background of the individual user, and may change as the user gains more experience. The naming of the parameters that is set out here is a reasonable starting point for contemporary English-speaking users with a reasonable familiarity of the workings of 4DML; it can be adapted for others as necessary.

6.3 Summary

When comparing 4DML with other generalised transformation frameworks, it can be seen that 4DML holds advantages with regard to visibility and lack of diffuseness, particularly when the designer is constrained to a text-editing environment, due to the brief-but-readable nature of 4DML models. The framework can also assist with closeness of mapping and reduce errors. However, there are some sources of error-proneness, and the brief nature of models, helpful though it can be, does not completely alleviate the need for thought and introductory explanation.

7 Implementing 4DML efficiently

This chapter discusses how 4DML can be implemented in a reasonably efficient manner using conventional hardware and programming languages. An algorithm is outlined that for many transformations has a time complexity of approximately $O(N)$ in the size of the input. While efficiency is by no means the overriding objective in the design of 4DML, it does benefit usability if transformations can be tested without incurring undue delay, since the transformation might be part of an incremental development cycle.

Chapter overview. The chapter begins by discussing the ideal complexity of transformation algorithms. It then explains how the complexity of a 4DML transformation depends on the complexity of the primitive "for each" operation, and presents two ways of implementing this efficiently; it goes on to show how input can be converted into a 4DML pointset in approximately $O(N)$ time. After a discussion of further ways of optimising the process, actual timings are given from the 4DML prototype, and the chapter concludes by describing possible future work on efficiency.

7.1 Objectives

Ideal complexity. Special-case transformation algorithms have varying complexities bounded by the nature of the transformation. For any transformation T, there is a mathematical lower bound on its complexity, such that it is not possible to construct a complete implementation of T with a lower complexity. Since the application of a generalised transformation algorithm to achieve T is in effect an implementation of T, it follows that the ideal complexity of a generalised transformation algorithm, when applied to any given transformation T, is no lower than the theoretical lower bound of T's complexity.

An ideal generalised transformation algorithm is one where, for *any* transformation T that lies in the scope of the generalised algorithm G, the complexity of the generalised algorithm applied to T (i.e. G_T) is no worse than the minimum complexity of T:

$$\forall T,\ O(G_T) = O(T)$$

This assumes that the parameter of G that is used to specify T (the *model* in the case of 4DML) is of constant size, so any overhead of parsing it will not dominate the overall time taken for large amounts of input data. It is also possible to stipulate that, if there are two or more ways of specifying T, an ideal generalised transformation algorithm should achieve the same minimum complexity regardless of which way T is specified; in practice this depends on the nature of those specifications and the decidability of their equivalence.

Complexity of matrix transposition. The minimum complexity of parsing and transposing a simple matrix is $O(R \times C)$ in the size of the matrix (rows × columns). (If the matrix were stored in a suitable data structure, it is possible to achieve transposition in $O(1)$ time, but that would not account for parsing and displaying it.) Since 4DML is frequently used for operations that resemble matrix transposition, one would expect these operations to be achievable in $O(N)$ time in the size of the input data, which would give $O(R \times C)$ since $N = RC$.

Effect of small input sizes. Big-O notation is used to reason about the limit as the input size tends towards infinity; other factors come into play when the inputs are small. Since small inputs are often the case, particularly when models are being prototyped incrementally (which is when a short response time really matters), it is beneficial to take this into account. Frequently it is possible to trade a higher complexity for a lower constant factor, and this pays off when the inputs are small.

For example, during the development of 4DML, two data structures were considered for storing the input; one kept the original input order and the other did not. Keeping the original input order would allow many transformations to be completed in $O(N)$ time, whereas not doing so would more frequently require a sorting operation and thus the complexity would be increased to $N \log N$. However, the constant of proportionality in the $N \log N$ part of the equation is very small when compared with the rest of it, and for small values of N it is far outweighed by the additional overhead of the first algorithm. Thus the $N \log N$ algorithm actually performs better until N is very large. In other words, if the first algorithm takes time αN for some constant α, and the second takes $\beta N + \gamma N \log N$ for some constants β and γ, then

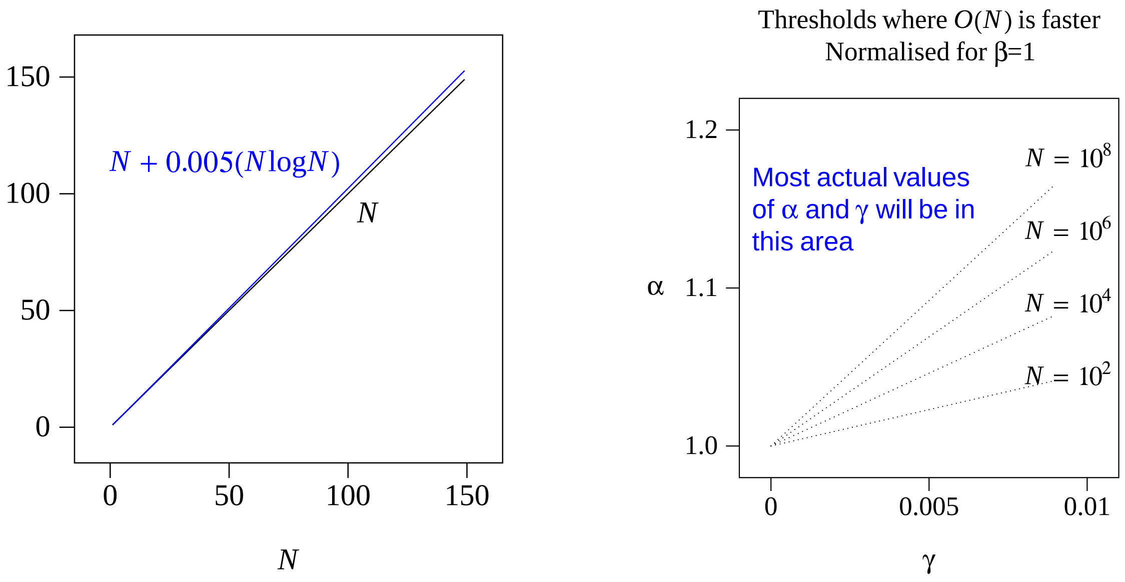
$$N < \exp(\frac{\alpha - \beta}{\gamma}) \Rightarrow \beta N + \gamma N \log N \ < \ \alpha N$$

As $\gamma \to 0$, the above threshold for N increases exponentially. Further, if $\gamma \ll \beta$, the time taken by the $O(N \log N)$ algorithm is negligibly more than βN for small values of N (Figure 7.1a). Consequently, if small input sizes are frequent (as they are) then it is beneficial to implement the $O(N \log N)$ algorithm.

If *both* algorithms are implemented, it is possible to achieve an $O(N)$-like algorithm with a smaller constant of proportionality when the input is small, by switching from one algorithm to the other once the input size surpasses a threshold. However, the pay-offs would hardly ever justify the development time, since the threshold increases so rapidly (Figure 7.1b).

7.2 The for-each operation and its effect on complexity

As described in Section 4.3, the 4DML "transformation by model" algorithm works by performing a top-down traversal of the model, reading off relevant parts of the input as it does so. The input is divided into subsets, normally with each subset corresponding to an instance of an element that is named in the model. Let this operation be called *for-each*, as in "for each element that matches some criteria, do the following. . . " Then most 4DML models are formed of nested for-each constructs.



(a) A small $N \log N$ term is barely perceptable

(b) Implementing $O(N)$ is rarely worthwhile

Figure 7.1: The effect of small input sizes

Dependence of overall transformation complexity on for-each complexity. Let the complexity of the for-each operation be denoted by the function $F(N)$, where N is proportional to the size of the subset of the data that this particular for-each operation is working with.

Models are built up from for-each operations. Two points may be observed:

1. A constant number of juxtaposed for-each operations is of the same complexity as for-each: $O(kF(N)) = O(F(N))$
2. If $F(N)$ is linear, then a constant number of *nested* for-each operations is of the same complexity as for-each.

The latter point needs some explanation. Suppose a for-each operation generates s subsets, and for each of those subsets a function of complexity $G(N)$ is executed. Since the average size of each subset will not normally exceed N/s (since there is no duplication, unless unusual parameters are in use), the overall complexity will be $O(F(N) + sG(\frac{N}{s}))$ if G is linear. If the function of complexity $G(N)$ is itself a nested for-each, or a constant number of such, then we can prove the above assertion by induction:

Base case. If there is no nesting, then $G(N) = 0$; substituting this into $O(F(N) + sG(\frac{N}{s}))$ gives $O(F(N))$.

Inductive step. We add one level of nesting to an existing nested structure that is $O(F(N))$, and prove that the result is also $O(F(N))$. The complexity of the existing nested structure corresponds to the G in $F(N) + sG(\frac{N}{s})$. If $O(G(N)) = O(F(N))$, then $O(F(N) + sG(\frac{N}{s})) = O(F(N) + sF(\frac{N}{s}))$. If $F(N)$ is linear, then F is distributive—$F(a) + F(b) = F(a + b)$. Consequently, $F(N) + sF(\frac{N}{s}) = F(N + s\frac{N}{s}) = F(2N)$ so $O(F(N) + sG(\frac{N}{s})) = O(F(N))$.

Hence, an implementation of for-each of $O(N)$ will result in a transformation system that can perform matrix transpositions and similar operations in $O(N)$ time—the minimum possible complexity if parsing and display is taken into account.

Recursive models. The above analysis assumes that the model is constant and does not use any special parameters such as recursion. Conditions are not always so ideal. Consider for example the effect of the following recursive model on an arbitrary tree of elements named `E`:

```
E/E call=E
```

This model will traverse the structure of the tree without producing any output (see page 66). If each for-each operation is of $O(N)$, then in the worst case, with an extremely unbalanced tree, this model will be of complexity $N+(N-1)+(N-2)+\ldots = \frac{1}{2}(N^2+N) = O(N^2)$. This is cause for concern, since mathematical expressions are essentially arbitrary recursive structures, and the models used to transform them follow the same outline as the above simple recursive model (let `E` stand for Expression). Further, the ideal complexity of many transformations of mathematical notation is $O(N)$, since these transformations usually involve straightforward substitution. 4DML would perform badly if it took $O(N^2)$ time for a substitution.

This situation can be improved if the implementation of for-each, and the data structures that it uses, are such that for-each is $O(1)$ when the element it is searching for is at the top level. Even if for-each is still $O(N)$ overall, improving the efficiency of top-level divisions would result in better performance for recursive models. Note that a complexity of less than $O(N)$ can only be achieved if for-each returns references to parts of its input, otherwise the overhead of copying the subset from the input would itself be $O(N)$. This will be discussed further in Section 7.3.2.

Clarification of N. Thus far it has been assumed that N is proportional to the size of the data, or to the amount of markup present. N is more accurately the number of four-dimensional points. Normally, these are proportional to each other—more data, or more markup, makes for a correspondingly greater number of four-dimensional points. However, it is worth noting that adding one more layer of markup might generate a large number of extra points even when the markup is simple—enclosing a document in an element named `document` will generate as many points as there are items of data, each representing the fact that this particular item of data is part of the `document`. If the data is not represented as a set of four-dimensional points, but is represented more compactly, then adding a `document` element might not increase its size so much.

For the purposes of analysis, this is rarely significant, because most large documents have a more-or-less constant number of layers of markup above them. Adding to them implies either adding more data with the same depth of markup (which increases N proportionally) or adding analytical markup over the whole data (which also increases

N proportionally). So if some algorithm has a complexity of $O(F(N))$ in the number of four-dimensional points, then in most cases it is reasonable to think of it as having the same complexity in the size of the data or the amount of markup, although the constant of proportionality will vary across different types of data and different models. As this variation is proportional to such things as the depth of the markup or model (which is normally small in comparison with the size of a large amount of data), it should not be excessive.

7.3 Two for-each algorithms

These algorithms are approximately $O(N)$, although there is a small additional $N \log N$ cost due to a sort operation. This additional cost is insignificant in most cases, as was previously explained (Figure 7.1).

7.3.1 Points in a random order

This algorithm can be used to conduct a for-each operation on a set of four-dimensional points, regardless of the order in which the points are stored. It is reminiscent of bucket sort, and has two stages:

1. Determine what the subsets ("buckets") are;
2. Copy the points into the subsets.

Each of these stages involves a pass over the data, which is $O(N)$. Additionally, the subsets need to be sorted into order, which involves a small additional $N \log N$ cost—the number of subsets is usually much smaller than N, and the sort itself is very fast because it is essentially sorting a list of integers (position numbers).

Determining the subsets. Pass through the points and identify the subset of them that match the specified element name (if a name has been specified). Of these, keep only the ones of minimum depth.

Build a hash table, the keys of which are references to items of data, and the values are element names and positions. (It is not necessary to hash the data itself, only the references to it.) Populate the hash table by going through the points that have been selected, taking a reference and an (element, position) pair from each point. The hash table now maps data references onto positions and element names. (Normally only positions are necessary, but element names are also included, to handle the case where they are not specified, such as when matching against wildcards—it is possible that two or more elements share the same position number at the same depth, in which case they will need to be differentiated.)

Copying the points into the subsets. Pass through all the points and, using the above hash table, examine the data that is referenced by each point and determine which subset it should be in. (Create another hash table to build up the subsets themselves—this will map positions and element names onto sets of points.) This works because each point refers to an item of data that is also referenced by one of the points of its containing markup that has been used to determine the subsets.

Finally, the subsets are converted into a list and sorted by position number.

Some of the points will not appear in any subset; it is useful to save these so that they can be added to the external stack (see Chapter 4). Otherwise, when constructing the external stack it would be necessary to pass through the points again to determine which ones have not been used.

7.3.2 Points in a more complex structure

As suggested in Section 7.2, it can be advantageous if for-each is less than $O(N)$ complexity for top-level operations on deep structures, i.e. adding more branches at deep levels does not affect operations at higher levels. This is the case for conventional tree data structures; however, it should still be considered that the elements that for-each deals with are not *necessarily* those at the very top level, and also that 4DML can represent more than one hierarchy over the same data.

This algorithm relies on the points being stored in lists. Each list is associated with an item of data, and it contains all the points that reference that item of data (the reference can therefore be omitted as no longer necessary); the points are stored in increasing order of depth. It is important that these lists are implemented as singly-linked lists (not doubly-linked lists or vectors), since the algorithm will create new lists that point part-way into existing lists in order to avoid unnecessary copying. It is also important that the fourspaces involved are read-only (or at least copy-on-write), since any changes might inadvertently affect other fourspaces, given that references are shared. This precludes any optimisations that require in-place modification in other parts of the program.

Determining the subsets. Obtain a vector of list iterators, one for each of the lists, each iterator pointing to the head of its respective list. Examine the points referred to by all of the iterators, to test if any of them specify the element name that is being searched for. (Accept *any* element if it is a wildcard.) If none is found, advance all the iterators, dropping any that pass beyond the ends of their lists, and repeat. If all the iterators are dropped, the result is empty.

When some iterators refer to appropriate points, drop all the other iterators. Then create an empty hash table that maps positions and element names onto subsets (which in this case are going to be groups of lists)—this is similar to the second hash table in the previous algorithm.

Generating the subsets. For each of the iterators that has not been dropped, create a new list as follows: Copy all the items on the original list, up to but not including the item pointed to by the iterator. Then link directly to the item after the iterator. This behaviour might be modified by parameters; for example, if the element is not to be removed then no modification to the list is necessary and the only action necessary is to generate a new reference to the entire list; alternatively if `children-only` is in force then the reference will refer to the list item after the iterator.

Each new list is placed into its appropriate subset, according to the details of the point referred to by the iterator. Then the subsets are sorted as before. The entry on the external stack consists of any lists whose iterators had been dropped.

7.3.3 Avoiding the sort operations

It is possible to avoid the sort operations altogether, at the expense of more overhead, which corresponds to a higher constant of proportionality in the $O(N)$ complexity. As discussed earlier (Figure 7.1), this is rarely worthwhile, but for completeness it is described here.

The objective is to preserve the original order of the input data, so that it does not have to be reconstructed by sorting. Hence, the first algorithm will work with ordered lists of points, rather than sets of points where ordering is insignificant; the second algorithm works with ordered lists of lists, rather than groups of lists where ordering is insignificant (although the ordering *within* the lists is always significant). In this case sorting is not necessary because the buckets will always be generated in ascending order if the input is normal XML or MML.

The additional overhead is generated by the fact that the subsets need to be maintained in order, rather than in a hash table (which does not preserve order). The hash table that maps positions and element names onto subsets can still be used for fast lookup during the for-each algorithms, but this time the hash table is acting as an index into the ordered list, and both of them must be maintained together. It is not possible to obtain fast access without a hash table by using a vector, because the position numbers are not necessarily contiguous, and element names can also be part of the key.

Since a sort operation with integer comparison is usually implemented as a very low-level library function, it is likely to take less time than the additional overhead described above. Even if the entire algorithm were implemented at a similarly low level, it would be a challenge to make the non-sorting implementation run faster on any case of reasonable length.

7.4 Parsing data into 4DML

This can be achieved in approximately $O(N)$ time, as this section will show.

Consider the case where the input is derived from XML, and an XML parser is available that produces SAX-like events, such as "begin element" and "end element", interspersed with the data. The algorithm works equally well when the data is taken from other sources, such as 4DML's Matrix Markup Language (MML), which can either be parsed directly into SAX-like events or converted into XML to be parsed by 4DML's XML parser.

XML technicalities. Some special considerations are necessary when representing XML in particular, which do not apply to every possible hierarchical markup system:

1. XML allows an element to contain cdata (character data) that is interspersed with other child elements. 4DML does not track the position of data, only of elements, since different sets of markup could completely change the ordering. For example, the following is valid in XML:

```
<P> d1 <A>...</A> d2 </P>
```

However, if this were represented in the four-dimensional point-set, it would not be possible to recover the relative positions of `d1` and `d2`, since positioning information is only associated with elements, not data. It therefore becomes necessary to add extra markup so that elements do not mix data with other child elements, as in:

```
<P> <cdata>d1</cdata> <A>...</A> <cdata>d2</cdata> </P>
```

When linearly parsing XML without look-ahead, the point representing the first `cdata` element may be added retrospectively once the `A` is encountered, or the additional markup could be introduced only over `d2` if we assume when processing the fourspace that any data occurs before other child elements. Alternatively, `cdata` elements may be added over *all* XML cdata, and unnecessary ones removed.

The actual 4DML prototype uses the empty string rather than a reserved word like `cdata`; these examples use `cdata` for clarity (if "`</>`" were legal then it would be ambiguous).

2. XML attributes are represented as children of a child element named `!attributes` (not valid in XML) which does not disturb the position numbers of the other children.

3. XML allows empty elements, which contain no data. Since the representation of an element requires it to have a scope over some data, it is necessary to represent the scope of an empty element as an empty string ϵ, which is still uniquified so that different empty elements can be differentiated by their scopes—as with other data, a new instance of ϵ is created for each empty element, and instances are uniquely referenced.

Conversion from XML. The conversion from XML is achieved by linearly traversing the XML document while maintaining a stack of the elements that are currently open. This element stack is represented as a three-dimensional point-set, the dimensions being element name, position, and depth—the first three of the four dimensions in 4DML. For example, if the parser has so far read:

```
<P> <A>...</A> <EXPR> <TERM>
```

then its internal state will be:

```
[(’P’,1,1) (’EXPR’,2,2) (’TERM’,1,3)]
```

When some data is read, it is uniquified and used as the fourth co-ordinate of every point in the above point-set, the resulting four-dimensional points being added to the fourspace. Thus the appearance of the string `pi` after the above might result in the following points being generated:

```
[(’P’,1,1,(’pi’,71)) (’EXPR’,2,2,(’pi’,71))
(’TERM’,1,3,(’pi’,71)) (”,1,4,(’pi’,71))]
```

where the last point represents the `cdata` element that was introduced as described above.

The value of the position dimension is reset to 1 when the depth increases. The position counter is *not* incremented after creating an `!attributes` element (see above), as this would disturb the position numbering.

Complexity. The maintenance of the stack is $O(1)$ for each operation, and the copying to the fourspace is $O(S)$ where S is the size of the stack. In the worst case, with very deeply-nested markup, the complexity in both time (parsing time) and space (number of 4DML points generated) is $O(N^2)$ in the length of the markup—consider the case when some constant fraction f $(0 \leq f < 1)$ of the XML data is adding a large number of items to the stack (so $S \propto fN$), which the rest of the XML data then uses to generate many fourspace points—the complexity is thus $O(fN + S(1-f)N) = O(fN + (fN)(1-f)N) = O(N^2)$. However, most actual data for which 4DML is well-suited is more likely to have a constant nesting depth d that does not increase with N; in this case the complexity is at worst $O(dN)$, which is $O(N)$.

Extensions for overlapping markup. Since 4DML's Matrix Markup Language (MML) allows the representation of multiple, overlapping markup systems while XML does not, an algorithm to parse XML into 4DML should ideally be exensible to cope with this overlapping markup without increasing its complexity. The above algorithm does have this property; it is sufficient to extend XML with the following two additional commands, each of which can be performed in constant time:

1. Increment the position number of a higher-level element (an element further up the tree), given its element name, without changing any of the markup that has been opened since that point. This can be performed in constant time (or time that is proportional to the length of the element name) if a suitable hashtable is available to find the element quickly.

2. As above, but reset the position number to an arbitrary value instead of incrementing it.

Clusters of several such commands can be given at once. User-defined operators in MML will trivially map to these commands.

7.5 Further optimisations

When implementing 4DML, the following further optimisations can increase the speed by a constant factor in some circumstances.

7.5.1 Pre-parsing the model

Since 4DML models are normally fairly small, it is possible to make their internal representations many times larger in order to avoid having to repeatedly parse and/or interpret them during a transformation with many iterations. Each node in a model's parse tree can be associated with additional variables, such as boolean values indicating the presence or absence of certain attributes, in a manner reminiscent of the microcode in early CISC processors.

When the implementation language is an interpreted language such as Python, the gain in speed is not as much as it could be, since every variable access involves a string lookup anyway. However, there is a non-zero improvement in pre-parsing and interpreting the model, and it also makes the source code more maintainable by separating the interpretive logic from the executive code.

7.5.2 Caching the results of for-each

Some models are repetitive in the way that they request for-each operations. Consider for example the following, which might be used to write out a song's verses and chorus in poetic form, with the chorus in its conventional position after the first verse:

```
verse number=1, chorus, verse start-at=2
```

In both instances, `verse` will result in a for-each operation to group the data by verse. Since for-each is an operation similar to bucket sort and is approximately $O(N)$, little is to be gained by modifying the core of the for-each algorithm to take account of the parameter `number=1`; instead, the entire list of subsets (verses) is returned and only its first element is processed (if any—see Section 6.2.2). Then the second instance of `verse` in the model will fetch precisely the same list, and process all other elements. Clearly, some time can be gained by keeping a reference to the previously-generated list and re-using that.

This assumes that the lists returned are not modified; otherwise, the overhead of taking separate copies of them for the cache can outweigh the cache's benefit. This in turn precludes any optimisations that require in-place modification of the lists or subsets. In practice that is not a great restriction.

It would seem that problems would arise when memory is limited, meaning that the cache requires logic to discard data that is not likely to be re-used soon. The overhead of such logic would take away from the benefits of the cache. In practice, such logic is not needed with small models, provided that there is one cache associated with each valid fourspace that has been the subject of a for-each operation, and that the memory consumed by the cache is freed at the same time as that consumed by the fourspace.

It is also possible to inspect models in advance and attempt to predict how they re-use for-each operations. Such prediction needs to be coded carefully if it is to cope with deeply-nested repetitions, perhaps in subroutines. Experiments showed that such efforts give diminishing returns.

7.5.3 Multithreading and parallel processing

Each for-each operation generates a list of subsets, and then further operations are performed for each of those subsets. It is not difficult to process several subsets in parallel by using thread-level parallelism if the system supports it, and then collate the output.

Relevance. Some systems will automatically parallelise multithreaded applications by running the threads on different processors (as in SMP), or on different pipelines within the same processor (as in Intel's "hyper-threading"); it is also possible to design processors that perform low-overhead context switches to other threads in order to utilise otherwise-idle time when fetching from higher-latency memory [73] and this is useful when the data will not entirely fit in the cache. If 4DML uses multiple threads then these developments can be taken advantage of, provided that the threads are true system-level threads and are not emulated by a user-level library or interpreter.

Limiting thread generation. Generally, a thread-aware architecture will have some maximum number of threads $T_{\max}$, where the addition of further threads will not improve performance and may degrade it. In the case of an architecture that is not thread-aware, $T_{\max} = 1$. 4DML should avoid running more than $T_{\max}$ threads at any one time; this can be achieved by stipulating that, after a for-each operation that generates s subsets, the number of new threads to generate is $\min(s-1, T_{\max} - T)$, where T is the current number of threads. Then the number of subsets to assign to each thread (including the current thread) is $\left\lfloor \frac{s}{N+1} \right\rfloor$ where N is the number of new threads that was generated; since this may not account for all the subsets due to rounding, any remaining subsets can be distributed to the first few threads (one each).

Overhead. The creation of new threads carries an overhead, and it is only worthwhile if the time saved by parallelisation exceeds that of thread creation. Hence it is rarely worthwhile to create a new thread to process a very small amount of data.

The above method of limiting thread generation has a side effect of generating most threads early on, where the size of data in each subset is likely to be large. However, it is possible that threads will be generated to process very small amounts of data, particularly when previous threads have finished; therefore it is advisable to modify the above approach so that only subsets larger than a given size are considered for threading; any that are smaller are processed by the current thread. The threshold size will depend on the architecture.

Implementation. The Python implementation of 4DML was made multi-threaded and run on Jython, the Java version of Python, on an SMP machine with two AMD CPUs. It was necessary to use Jython because all other Python systems (at the time of writing) use a "global interpreter lock" to serialise access to Python objects, which is not suitable for true parallel processing (the runtime is some 30% longer than the non-threaded version).

Under Java 1.2.2, running with $T_{\max} = 2$ resulted in 35–40% more CPU time than with $T_{\max} = 1$ (mainly due to inefficiencies in Jython/Java while threads were running—the overhead of thread creation was a very small part of that). Since the CPU time was divided between two CPUs, one might expect the overall runtime to be reduced by $100\% - \frac{140\%}{2} = 30\%$, but actually the figure varied between 5–10%—each CPU was only running at around 80% utilisation due to the overheads of sharing memory between CPUs; there can be contention on the memory bus even when the areas of memory being accessed are different.

Similar results were obtained on a SPARC machine with 4 CPUs (Figure 7.2). A 2-CPU Intel Xeon machine (where each CPU has two pipelines, allowing four parallel threads in

$T_{\max}$	Increase in CPU time	Reduction in overall runtime
2	73%	0.5%
3	108%	11.8%
4	119%	16.7%

Figure 7.2: Performance of parallel processing on a SPARC SMP machine (figures relative to $T_{\max} = 1$)

$T_{\max}$	Increase in CPU time	Reduction in overall runtime
2	62%	−4.5%
3	126%	−5.6%
4	205%	−24.0%

Figure 7.3: Performance of parallel processing on an Intel Xeon machine (figures relative to $T_{\max} = 1$)

total) performed very badly (Figure 7.3), probably due to the memory-oriented nature of the code.

Distributed processing. Threads can be sent to the nodes of a distributed computer, such as a Beowulf cluster [8]. In this case the memory is not shared—the distributed system is composed of several discrete computers connected across a network. This means that any data that is to be processed on other nodes must first be transmitted across the network—the overhead of setting up a new thread will vary with the size of the subset(s) assigned to it. Whether such a processing model is worthwhile will depend on the performance ratio between the processors and the networking—the faster the processors are in relation to the network, the more likely the network is to dominate. Most actual applications of 4DML are unlikely to benefit from distributed processing.

Finer-grain parallelism. If a part of a 4DML model is of the form `A, B, C,...` then it is possible to run these in parallel so long as there are no data dependencies between them. In other words, the output from all the threads processing `A` does not have to be completely collated before `B` is started; `B` can start as soon as there are available threads. This leads to complications.

One example of a data dependency is a "case" block with a "default" condition (e.g. `value` and `other-values`)—the default part needs information about whether or not the others matched. This precludes at least some of the parallelism. Another data-dependency is the cache of previous for-each results—if this is to remain useful then some complex locking semantics would have to be implemented, which in turn would mean that some of the threads can be blocked and it may be beneficial to create slightly more than $T_{\max}$ threads to compensate.

It is also possible to parallelise for-each itself, over different parts of the data, but collating the output of the parallel threads would be more complex.

Limitations. If 4DML is extended with further parameters then some of these parameters might introduce dependencies across iterations that would preclude some of the

Task	Time (secs)
Weather report (Figure 5.2)	2.79
Customised notation for Chinese (Figure 5.11)	1.12
Distributed music encoding (Figure 5.16)	17.53
Aspect-oriented composition (Figure 5.18)	7.17
Japanese koto notation (Figure 5.23)	7.76
Diagram (Figure 5.35)	3.24
Diagram subset (Figure 5.37)	1.87

Figure 7.4: Running times of selected transformation tasks

parallelism, hence making it more complex. In this case, it seems likely that better efficiency can usually be achieved by selectively parallelising only the parts of the model that do not have such parameters.

7.6 Practical results

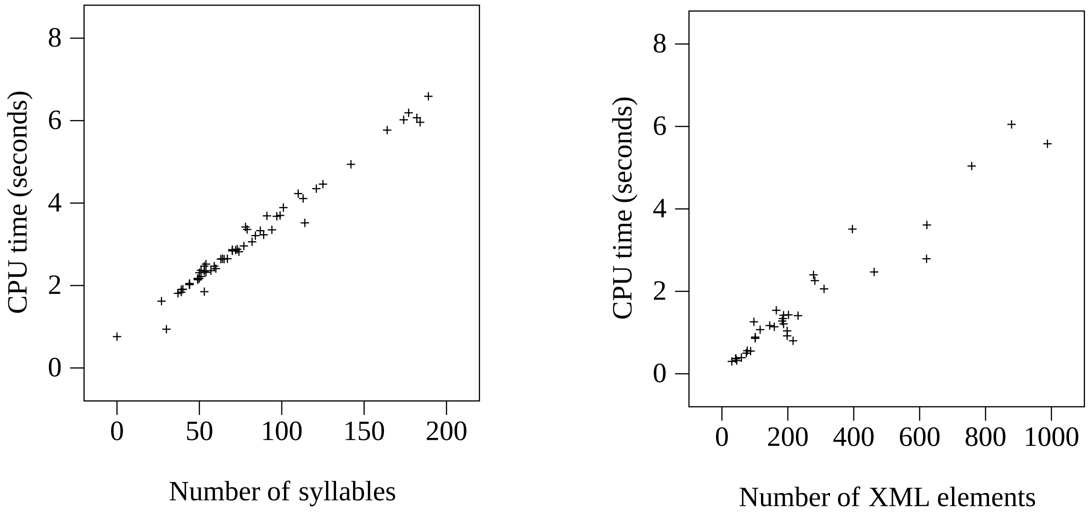
By way of example, this section presents some actual timings from the 4DML prototype. The prototype was written in the Python programming language [70] for ease of maintenance. Although Python introduces some overhead as an interpreted language, it is still possible to devise algorithms that are efficient in that their worst-case complexity is no higher than needed; if the program were re-written in a faster language (such as C++) then the resulting speedup would be a constant factor.

Typical running time. Tests were conducted on a desktop PC (700MHz CPU, 128M RAM, Linux) with Python version 2.2 running precompiled bytecode. The prototype completed most small transformation tasks within a few seconds (Figure 7.4)—times shown include parsing and transformation but exclude non-4DML tasks such as running LaTeX.

Complexity. Figure 7.5 shows how differing input sizes affect the processing time. Figure 7.5a was generated by transforming the collection of Chinese documents mentioned in Section 5.4, and shows a good approximation to $O(N)$ in the size of the input (as measured by the number of syllables in the document). Figure 7.5b was generated by transforming a collection of MathML documents, which was obtained by converting a large sample of LaTeX files to MathML and then isolating those with significant quantities of mathematics. It is still roughly $O(N)$, but there is more deviation because there is greater variability in the structure of the input.

7.7 Scope for future work

The following items of work on the efficiency of 4DML could potentially make research projects in themselves:



(a) producing customised language-learning materials (Section 5.4)

(b) converting mathematics to speech (Section 5.3)

Figure 7.5: Effect of input size on time taken

1. Develop an optimising compiler that translates 4DML models into optimal low-level code. Once compiled, a model can then be used on new data very quickly, and it would be feasible to use 4DML as a production development tool even when optimised code is required. (Optimisation can be for speed, code size, memory requirements, use of particular embedded hardware, etc.)

2. In an interactive application, it is possible that a transformation will be run repeatedly, with small changes in the data and/or model between runs. It should be possible to take advantage of previous runs in order to save calculations, in other words, to run 4DML incrementally. That way, small edits in large documents will not give unnecessarily long response times before appropriate parts of the display are updated. It might also be possible to determine which parts of the transformation are most important for immediate display, and to perform these first.

7.8 Summary

This chapter discussed how 4DML can be implemented efficiently. Two algorithms were presented for the "for-each" operation, which is the core of a 4DML transformation. It was shown that many transformations can be accomplished in approximately $O(N)$ time in the size of the input, and this was confirmed by actual timings. The chapter also discussed further optimisations, including the exploitation of parallel processing, and future possibilities for improving efficiency.

8 Conclusion and further work

As was outlined in the introduction (Section 1.5), the work described in this dissertation aimed to develop a generalised transformation framework that can be applied to many different transformations. This has been achieved and examples of such use are shown in Chapter 5.

8.1 Review of aims and objectives

Section 1.5 listed three high-level goals for the transformation framework:

1. The method of programming the framework for a particular transformation task should encourage, wherever possible, a consideration of the notations themselves rather than the algorithmic methods for their transformation.

2. The framework should encourage the prototyping and customisation of new transformation tasks—this should require as little effort as possible.

3. The framework should integrate with other transformation systems, particularly typesetting systems, so as to take advantage of the enormous amount of research and development that has already been done in this area.

4. The framework should further facilitate the creation of new notations on demand in order to address unusual educational difficulties or special circumstances.

Each of these goals will now be discussed in turn.

8.1.1 Encourage consideration of notations themselves

As can be seen from the description of 4DML in Chapter 4 and evaluation in Chapter 6, the framework fulfils the objective of encouraging a consideration of the notations themselves rather than the algorithmic methods for their transformation. It allows the user to specify the *structure of the desired result* in a fairly concise manner, without focusing on algorithms or transformation rules as is normally the case.

This is not to say that a 4DML model does not specify an algorithm. It *does* specify an algorithm, because the behaviour of models is well-defined—4DML does not "guess" the user's intentions, but uses the model as a guide to reading and re-structuring the input in a distinct manner as was shown in Chapter 4. Authors of models know what they can expect 4DML to do; it is not necessary to "train" the system with many examples, and there is no uncertainty as to whether or not it will "understand". It is a rigorous programming language that could be used in safety-critical applications. However, the *form* of 4DML models is such that it puts the emphasis on the desired output notation rather than on the process. Hence 4DML encourages a consideration of the notations themselves without introducing the overhead of training by example.

8.1.2 Encourage prototyping and customisation of new transformation tasks

4DML's primary contribution is probably the brief-but-readable nature of its models, which helps the rapid prototyping of transformations. If transformations are prototyped rapidly then they can be prototyped experimentally or in situations where resources are limited, and thus a greater number of notational difficulties can be addressed by using conversion—without 4DML's facilities for rapid prototyping, the development would be prohibitive or at least discouraging.

8.1.3 Integrate with other transformation systems

Practically every example in Chapter 5 shows 4DML integrating with other transformation systems, particularly typesetting systems. Hence 4DML takes advantage of the extensive work that has previously been achieved in this area. It is also possible to combine a 4DML transformation with arbitrary program code, as shown in Section 4.8.1; this makes it easier to utilise existing work when experimenting with 4DML.

8.1.4 Facilitate creation of new notations to address unusual difficulties

The value of 4DML for creating new notations to address difficulties experienced by individual users was particularly demonstrated in Section 5.4—both the output in Figure 5.11 and the input in Figure 5.10 are custom notations to assist language learning for users with special needs. The distributed music encoding and aspect-oriented composition in Section 5.5.1 and the customisation of diagrams in Section 5.7 provide further illustrations. In each case the transformations were prototyped quickly, with the focus being on the notations themselves. This would have been much more difficult without 4DML.

8.2 Further work

Virtually all information-society applications involve notations, and the transformation of these between different versions is a component part of universal access, since it can help to cater for special needs and for differing tasks and environments. Tools that support the programming of such transformations facilitate the creation of new notations on demand and the implementation of universal design. Some possibilities for further work in this area are described below.

8.2.1 Interactivity and collaboration

Speed. As previously noted (Section 7.7), if an interactive editor incorporates a realtime notational transformation then it is useful to implement this in such a way that it will run quickly, either by making the whole process fast or by implementing an incremental algorithm so that only what has changed need be processed. The 4DML prototype can be improved in this respect as explained in Section 7.7.

Collaboration and shared pointing. One of the primary problems with the conversion of notations as a means of addressing special educational needs is that it can complicate collaboration—if one person is using a printed notation and another is using Braille, for example, then they cannot say to each other "look at this symbol here" because they cannot use each other's notation; instead they must refer to some method of indexing that is common across the documents ("look at the second part of equation 23" or "look at the third note in bar 51"). However, if sufficient hardware is available so that the documents can be displayed using interactive display devices that allow pointing, then it may be possible to arrange for the pointing to something on one display to highlight (in some way) the equivalent place on the other(s), regardless of whether or not the notation is the same. Such pointing might be supported with hardware such as:

- a touch-sensitive refreshable Braille display—most refreshable Braille displays have keys attached to every cell of the display to facilitate pointing;
- a touch-sensitive or pen-based visual display;
- a video-augmented environment such as the DigitalDesk [61], which can superimpose a projection on a paper document;
- a conventional visual display with a mouse.

The transformation algorithm would keep track of which part(s) of the input affected which part(s) of the output, so that the positions on the various displays can be matched up. The use of a generalised transformation system has the advantage that it enables this to be implemented simultaneously for a wide range of transformations in many different notations. However, it may be necessary to integrate the transformation system with the typesetting and display systems at least to some degree; to this end, the use of a single generalised typesetting system for all the notations in question would be helpful.

In the long term, it might even be possible for some of the pointing to be done automatically with the aid of a speech recognition or musical-score following system. If the hardware becomes sufficiently portable and economical, one can envisage such applications as an orchestral workshop in which each musician reads the music from a touch-sensitive display, and touching the display to indicate a position will highlight the same position on the parts of all the other players even if they are using different notations.

Passive navigation. One of the problems raised by shared pointing is passive navigation—if one person points somewhere, then the other viewers must be guided from their present focus of interest to the new one, which might not fall within their current display area or the part of it that they are concentrating on. In some cases the viewers may require a certain amount of time to re-orientate themselves in the new position, if it is so far away that the pointer cannot be moved there from its previous position in a continuous sweep at a slow enough speed without taking too long.

In the case of the pointer suddenly jumping to a completely new context, if it is visual then it can be made to stand out as much as necessary; if it is tactile or auditory then this will be more difficult and some amount of time must be allowed for the viewer to take in the new context. Symbols that are close together in one notation might have equivalents that are further away in another, and some displays might not be able to show so much data as others, so this may become an issue even when the collaborator doing the pointing does not intend it.

One strategy that might help to alleviate the problem is to split the display so that it shows two or more different parts of the document at the same time, determined automatically from a record of the areas that have recently been highlighted more often; frequent navigation between these parts can then take place without altering the scope of the display. If the display is so small that it cannot reasonably be split, then it might at least be possible to make the context switch in such a way that the pointer does not move relative to the display.

If scrolling is possible then there should be some decision about how much the display can scroll and how much the pointer should move in relation to the display. This might depend on each user.

8.2.2 Further support of unusual notations

As explained in the introduction to this thesis, the use of unusual notation systems is an under-explored option in the relief of educational difficulties, primarily because present-day computer software does not offer enough support for the creation of new notations. 4DML is an example of a system that was designed to address this by facilitating the transformation of notations. There are other obstacles that can be addressed, such as:

1. When a non-standard or customised notation is in use, it is not always feasible to devise and implement *typesetting constraints* to the same degree as has been done with the established notations. Typesetting is an art that has been meticulously developed over the centuries, as shown by the intricate complexity of the rules of best practice that are encoded in TEX [44] and its auxiliary packages as well as music engraving programs like *Sibelius* [24] and GNU Lilypond [21]. These rules make for documents that are easier to read and to navigate, allowing the users of the document to devote more of their mental effort to the activity for which the document is an aid. However, computer displays and printouts of new or minority notations are rarely optimal in their ease of reading.

2. Users may require assistance in the *design* of their new notations; they may be discouraged from doing so if they do not understand the underlying principles, such as the Gestalt theory of visual organisation and the large body of psychological research in the areas of visual search and short-term memory.

   These principles are particularly relevant when the circumstances restrict what can be perceived concurrently, or the user has a restricted field of vision or an unstable gaze. For example, Figure 5.11 on page 90 was designed such that a reader with a restricted field of view has as much assistance as possible when “navigating” around the notation, and can rapidly recover from an accidental “jump” due to nystagmus or a similar condition.

However, not every user who has such special needs will be able to create such designs without assistance, and the notations designed by others might not meet their requirements precisely.

These problems are related, since the design of a notation (and the special need that it addresses) will frequently affect the typesetting constraints.

A research project to address these issues would include an investigation into the established rules of best typesetting practice, with a view to generalising them so that they can be applied to new notations and optimised for different special circumstances. Existing research can be built on to develop a formal methodology for the design of new notations to suit different circumstances, thus increasing the accessibility of a wide range of educational and cultural activities. 4DML would be an important tool in such a project.

8.3 Closing remarks

This dissertation argued that conversion between different forms of notations is an important part of universal access to scientific, educational and cultural activities. It exhibited a system for supporting such conversion, and demonstrated that this system and its methods significantly contribute to our ability to mitigate special difficulties by facilitating the use of alternative notations to address such difficulties. The system is also a useful research tool for supporting other notation-related activities.

Bibliography

[1] Anonymous ("Agent 33"). *AlterPage*. http://www2.crecon.com/agent33/alter.page.

[2] A. Aho, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[3] Jonathan Allen. Reading machines for the blind: The technical problems and the methods adopted for their solution. *IEEE Transactions on Audio and Electroacoustics*, 21(3):259–264, June 1973.

[4] Chieko Asakawa and Hironobu Takagi. Annotation-based transcoding for nonvisual web access. In *Proceedings of the Fourth International ACM Conference on Assistive Technologies ASSETS 2000*, pages 85–91, Nov 2000.

[5] Mikhail Auguston. RIGAL—a programming language for compiler writing. *Lecture Notes in Computer Science*, 502:529–564, 1991.

[6] Ira D. Baxter. DMS: practical code generation and enhancement by program transformation. In *Workshop on Generative Programming*, pages 19–20, 2002.

[7] David M. Beazley. PLY (Python Lex-Yacc). Available at http://systems.cs.uchicago.edu/ply/.

[8] Donald J. Becker, Thomas Sterling, Daniel Savarese, John E. Dorband, Udaya A. Ranawake, and Charles V. Packer. BEOWULF: A parallel workstation for scientific computation. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, 1995.

[9] Ron Ben-Natan. Web services in a pervasive computing environment. *WebSphere Developers Journal*, 1(9), Sep 2002. SYS-CON Publications, Inc., ISSN 1535–6914.

[10] Jon Louis Bentley. Programming pearls: Little languages. *Communications of the ACM*, 29(8):711–721, August 1986.

[11] Tim Berners-Lee. A Summary of the World Wide Web System. *ConneXions*, 6(7), July 1992.

[12] Peter Breuer and Jonathan Bowen. A prettier compiler-compiler: Generating higher order parsers in C. *Software—Practice and Experience*, 25(1):1263–1297, Nov 1995.

[13] Silas S. Brown and Peter Robinson. A World Wide Web mediator for users with low vision. In *ACM CHI 2001 Workshop No. 14*. `http://www.ics.forth.gr/proj/at-hci/chi2001/files/brown.pdf`.

[14] David Calderwood. Computer games for the blind. *Beebug*, 3(8):26, Jan/Feb 1985.

[15] World Wide Web Consortium. *Evaluation, Repair, and Transformation Tools for Web Content Accessibility*. `http://www.w3.org/WAI/ER/existingtools.html`.

[16] James R. Cordy, Charles D. Halpern, and Eric Promislow. TXL: A rapid prototyping system for programming language dialects. In *Proceedings of The International Conference of Computer Languages*, pages 280–285, Miami, FL, Oct 1988.

[17] Daniel Dardailler. *The ALT-server: An accessibility collaboration project proposal.* `http://www.w3.org/WAI/altserv.htm`.

[18] Tsuyoshi Ebina, Seiji Igi, and Teruhisa Miyake. Fast web by using updated content extraction and a bookmark facility. In *Proceedings of the Fourth International ACM Conference on Assistive Technologies ASSETS 2000*, pages 64–71, Nov 2000.

[19] Tzilla Elrad, Robert E. Filman, and Atef Bader. Aspect-oriented programming: Introduction. *Communications of the ACM*, 44(10):29–32, Oct 2001.

[20] Abraham Nemeth et al. *The Nemeth Braille Code for Mathematics and Science Notation.* American Printing House for the Blind, 1972.

[21] Han-Wen Nienhuys et al. *LilyPond — The GNU Project Music Typesetter.* `http://www.lilypond.org/`.

[22] Mark R. Boyns et al. *Muffin World Wide Web Filtering System.* `http://muffin.doit.org`.

[23] Paul Denisowski et al. Cedict project. `http://www.mandarintools.com/cedict.html`.

[24] Ben Finn and Jonathan Finn. *Sibelius: The Music Notation Software*, 2001. Sibelius Software Ltd, Cambridge, `http://www.sibelius-software.com/`.

[25] Office for Standards in Education. *Inspection Report—RNIB New College, Worcester*, page 37. Alexandra House, 33 Kingsway, London, WC2B 6SE, Oct 2000. Inspection number 223644.

[26] E. R. Gansner, S. C. North, and K. P. Vo. DAG—a program to draw directed graphs. *Software—Practice and Experience*, 17(1):1047–1062, 1988.

[27] T. R. G. Green and A. F. Blackwell. Design for usability using cognitive dimensions. Tutorial session at British Computer Society conference on Human Computer Interaction HCI'98, 1998.

[28] Peter Gregor and Alan F. Newell. An emperical investigation of ways in which some of the problems encountered by some dyslexics may be alleviated using computer techniques. In *Proceedings of the Fourth International ACM Conference on Assistive Technologies ASSETS 2000*, pages 85–91, Nov 2000.

[29] R. E. Griswold, J. F. Poage, and I. P. Polonsky. *The SNOBOL4 Programming Language*. Prentice-Hall, second edition, 1971.

[30] Dirk Hermsdorf, Henrike Gappa, and Michael Pieper. Webadapter: A prototype of a WWW-browser with new special needs adaptations. In *Proceedings of the 4th ERCIM Workshop on 'User Interfaces for All'*, number 8 in Long Papers: WWW Browsers for All, page 15. ERCIM, 1998.

[31] Anita W. Huang and Neel Sunderson. Aurora: A conceptual model for web-content adaptation to support the universal usability of web-based services. In *Proceedings of the 2000 International Conference on Intelligent User Interfaces*, Easy Access and The Web, pages 124–131, 2000.

[32] Scientific Computing Associates Inc. *Linda User's Guide and Reference Manual*, Dec 2001. 265 Church Street, New Haven, USA.

[33] Julie A. Jacko, Max A. Dixon, Robert H. Rosa, Jr., Ingrid U. Scott, and Charles J. Pappas. Visual profiles: A critical component of universal access. In *Proceedings of ACM CHI 99 Conference on Human Factors in Computing Systems*, volume 1 of *Profiles, Notes, and Surfaces*, pages 330–337, 1999.

[34] Julie A. Jacko and Andrew Sears. Designing interfaces for an overlooked user group: Considering the visual profiles of partially sighted users. In *Third Annual ACM Conference on Assistive Technologies*, pages 75–77, 1998.

[35] I. Jacobs, J. Gunderson, and E. Hansen (eds.). User agent accessibility guidelines 1.0. Technical report, W3C Recommendation, Dec 2002. `http://www.w3.org/TR/2002/REC-UAAG10-20021217/`.

[36] Patricia Johann and Eelco Visser. Warm fusion in Stratego: A case study in the generation of program transformation systems. *Annals of Mathematics and Artificial Intelligence*, 29(1–4):1–34, 2000.

[37] M. Jonge, E. Visser, and J. Visser. XT: a bundle of program transformation tools. *Electronic Notes in Theoretical Computer Science*, 44, 2001.

[38] Mitsuji Kadota. *Japanese Braille Tutorial*, Oct 1997. `http://buri.sfc.keio.ac.jp/access/arc/NetBraille/etc/brttrl.html`.

[39] A. Kennel, L. Perrochon, and A. Darvishi. WAB: World-wide web access for blind and visually impaired computer users. In *New Technologies in the Education of the Visually Handicapped, Paris, and ACM SIGCAPH Bulletin*, June 1996. `http://www.inf.ethz.ch/department/IS/ea/blinds/`.

[40] B. Kernighan and P. Plauger. *Software Tools*. Addison-Wesley, 1976.

[41] Jeffrey H. Kingston. The design and implementation of the Lout document formatting language. *Software—Practice and Experience*, 23:1001–1041, 1993.

[42] D. E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–146, 1968.

[43] Donald E. Knuth. Backus Normal form vs. Backus Naur form. *Communications of the ACM*, 7(12):735–736, December 1964.

[44] Donald E. Knuth. *The TEXbook*. Computers and Typesetting. Addison-Wesley, 1986.

[45] Didier Langolff, Nadine Jessel, and Danny Levy. MFB (music for the blind): A software able to transcribe and create musical scores into braille and to be used by blind persons. In *Proceedings of the 6th ERCIM Workshop on 'User Interfaces for All'*, number 17 in Short Papers, page 6. ERCIM, 2000.

[46] Harold J. Leavitt and Thomas L. Whisler. Management in the 1980s. *Harvard Business Review*, 36(41/1), Nov–Dec 1958. Also quoted in the Oxford English Dictionary, 2nd edition vol 7 p946, Oxford University Press 1989.

[47] Bruce R. Lewis. *BRL: A database-oriented language to embed in HTML and other markup*. `http://brl.sourceforge.net/`.

[48] S. Ludi. Are we addressing the right issues? Meeting the interface needs of computer users with low vision. In Simeon Keates, P. John Clarkson, Patrick Langdon, and Peter Robinson, editors, *Proceedings of the First Cambridge Workshop on Universal Access and Assistive Technology*, pages 9–12. Engineering Design Centre, Cambridge University Engineering Department, Mar 2002. Technical Report 117, ISSN 0963–5432.

[49] Bill McCann. *GOODFEEL Braille Music Translator*, Jun 1997. Dancing Dots Braille Music Technology, `http://www.dancingdots.com/`.

[50] Scott McPeak. Elkhound: A fast, practical GLR parser generator. Technical Report UCB/CSD-2-1214, University of California, Berkeley, Computer Science Division (EECS), University of California, Berkeley, California 94720, Dec 2002.

[51] Wayne Myers. *BETSIE (BBC Education Text to Speech Internet Enhancer)*. `http://www.bbc.co.uk/education/betsie/`.

[52] Elizabeth D. Mynatt and W. Keith Edwards. Mapping GUIs to auditory interfaces. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Audio and Asynchronous Services, pages 61–70, 1992.

[53] Cognitive Dimensions of Notations. T. R. G. Green. In Alistair Sutcliffe and Linda Macaulay, editors, *People and Computers V: Proceedings of the Fifth Conference of the British Computer Society*, pages 443–460. Cambridge University Press, Nov 1989.

[54] Ian J. Pitt and Alistair D. N. Edwards. Improving the usability of speech-based interfaces for blind users. In *Second Annual ACM Conference on Assistive Technologies*, Vision Impairments – II, pages 124–130, 1996.

[55] Sebastian Rahtz. PassiveTEX. Text Encoding Initiative. Available in most TEX distributions, 2003.

[56] T. V. Raman. *Audio System for Technical Readings*. PhD thesis, Cornell University, 1994.

[57] T. V. Raman. Emacspeak: a speech-enabling interface. *Dr. Dobb's Journal*, Sep 1997.

[58] Richard Rubinstein and Julian Feldman. A controller for a Braille terminal. *Communications of the ACM*, 15(9):841–842, September 1972.

[59] Peter Salus, editor. *Little Languages and Tools*, volume 3 of *Handbook of Programming Languages*. Macmillan Technical, first edition, 1998.

[60] Diomidis Spinellis. Unix tools as visual programming components in a GUI-builder environment. *Software—Practice and Experience*, 32(1):57–71, January 2002.

[61] James Quentin Stafford-Fraser. *Video-Augmented Environments*. PhD thesis, University of Cambridge, Feb 1996.

[62] Constantine Stephanidis. Aims and scope. *Universal Access in the Information Society*, 1(1):A4, 2001.

[63] C. Strachey. A general purpose macro generator. *Computer Journal*, 8(3):225 ff, 1965.

[64] Hironobu Takagi and Chieko Asakawa. Transcoding proxy for nonvisual web access. In *Proceedings of the Fourth International ACM Conference on Assistive Technologies ASSETS 2000*, pages 164–171, Nov 2000.

[65] Yoshiaki Takata, Takeshi Nakamura, and Hiroyuki Seki. Automatic accessibility guideline validation of XML documents based on a specification language. In Constantine Stephanidis, editor, *Proceedings of the 10th International Conference on Human-Computer Interaction (HCII 2003), Vol.4: Universal Access in HCI*, pages 1040–1044. Lawrence Erlbaum Associates, June 2003.

[66] Daniel Taupin, Ross Mitchell, and Andreas Egler. *MusiXTEX: Using TEX to write polyphonic or instrumental music*, Apr 1999. `ftp://ftp.gmd.de/music/musixtex/musixdoc.ps`.

[67] David Taylor and Christopher Harris. About nystagmus. Technical report, Nystagmus Network, 108c Warner Road, Camberwell, London, SE5 9HQ, UK, Sep 1999. `http://www.btinternet.com/~lynest/nystag.pdf`.

[68] NorKen Technologies. ProGrammar. `http://www.programmar.com/`.

[69] Jim Thatcher. Screen reader/2: Access to OS/2 and the graphical user interface. In *First Annual ACM Conference on Assistive Technologies*, Vision Impairments – I, pages 39–46, 1994.

[70] G. van Rossum. A tour of the Python language. In R. Ege, M. Singh, and B. Meyer, editors, *Proceedings. Technology of Object-Oriented Languages and Systems, TOOLS-23*, pages 370–. IEEE Computer Society Press, 1998. IEEE catalog number 97TB100221.

[71] G. C. Vanderheiden. Fundamental principles and priority setting for universal usability. In *Proceedings of the ACM Conference on Universal Usability (CUU)*, pages 32–38, Nov 2000.

[72] G. C. Vanderheiden. Why do we? Why can't we? Future perspectives and research directions. Closing Plenary Address of the SIG–CHI 2001 Conference, Apr 2001.

[73] Panit Watcharawitch and Simon Moore. JMA: the Java-multithreading architecture for embedded processors. In *Proceedings of the 20th International Conference on Computer Design (ICCD)*, pages 527–529. IEEE Computer Society, 2002.

[74] Jennifer Watts-Perotti and David D. Woods. How experienced users avoid getting lost in large display networks. *International Journal of Human-Computer Interaction*, 11(4):269–300, 1999. ISSN 1044–7318.

[75] G. Weber, D. Kochanek, C. Stephanidis, and G. Homatas. Access by blind people to interaction objects in MS Windows. In *Proceedings of the ECART 2 European Conference oon the Advancement of Rehabilitation Technology (Stockholm)*, page 2, May 1993.

[76] Stephen Wolfram. *The Mathematica Book*. Cambridge University Press, fourth edition, Apr 1999.

[77] World Wide Web Consortium. *XSL Transformations (XSLT) Version 1.0, W3C Recommendation*, Nov 1999. `http://www.w3.org/TR/1999/REC-xslt-19991116`.

[78] World Wide Web Consortium. *Extensible Markup Language (XML) Version 1.0 (Second Edition)*, Oct 2000. `http://www.w3c.org/TR/2000/REC-xml-20001006`.

[79] Ka-Ping Yee. *Definition of a Mediator*. `http://www.lfw.org/ping/mediator.html`.

[80] Ka-Ping Yee. *Shodouka*. `http://www.lfw.org/shodouka/`.

[81] Mary Zajicek, Chris Powell, and Chris Reeves. A web navigation tool for the blind. In *Third Annual ACM Conference on Assistive Technologies*, pages 204–206, 1998.

Index