A Knowledge-Based Environment for the Design and Analysis of Multidimensional Multirate Signal Processing Algorithms

A THESIS

Presented to

The Academic Faculty

By

Brian Lawrence Evans

In Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy in Electrical Engineering

Georgia Institute of Technology

June, 1993

A Knowledge-Based Environment for the Design and Analysis of Multidimensional Multirate Signal Processing Algorithms

Approved:

James H. McClellan, Chairman

Ronald W. Schafer

Russell M. Mersereau

Date approved by Chairman _____

To my parents

Acknowledgments

I would like to thank my parents for their encouragement, support, motivation, and drive to search for truth and pursue knowledge. My parents were great examples of the personal benefits of higher education. I would also like to thank my sisters, Melanie and Wendy, for their concern and support over the years.

I would like to thank my advisor and mentor, Dr. James McClellan. He has motivated me to reach my potential as researcher and allowed me to pursue the educational impact of my work. Dr. McClellan has my utmost respect.

I would like to thank the members of the reading committee— Dr. Russell Mersereau, Dr. Ronald Schafer, and Dr. James McClellan— and Dr. Alfred Andrew from the Mathematics Department for thoroughly reading the thesis. The *Mathematica* initiatives in the signal processing courses here at Georgia Tech would not have been possible without the *Mathematica*-based Calculus initiatives spearheaded by Dr. Andrew, Dr. Thomas Morley, and Dr. George Cain.

I would also like to thank the fifth member of my defense committee, Dr. Sudha Yalamanchilli. I have enjoyed the basketball games with and against him.

I would like to thank Dr. Barnwell and Dr. Schafer for establishing and maintaining an excellent signal processing program.

I would like to thank Dr. David Schwartz for suggesting the use of *Mathematica* as a platform upon which to build a symbolic signal processing environment. Later, *Mathematica* turned out to be useful in helping students learn signal processing theory.

I would like to thank Dr. James Wiltsie for sponsoring my research at the Georgia

Tech Research Institute (GTRI) and Dr. Chrystanos Papanicolopulos for hiring me at GTRI for two quarters after I completed my Master's degree.

I would like to thank the State of Georgia for my primary financial support as a research assistant. I would like to thank Dr. Sayle for supporting me as a teaching assistant to develop *Mathematica* courseware. The remainder of my funding came from the Joint Services Electronics Program under contract #DAAL-03-90-C-0004.

I would like to thank Dr. Jeffrey Froyd and Dr. Mark Yoder of the Rose-Hulman Institute of Technology for their feedback over the years. I have appreciated all the comments and suggestions by users of the signal processing packages and Notebooks.

I would like to thank two professors from the University of Queensland, Dr. Keith Matthews of the Mathematics Department and Dr. George Havas of the Computer Science Department, for their insights into Smith form decomposition algorithms.

I would like to express my appreciation for my closest friends Sam "Jan-Toua" Liu, "Fatma" Ayhan Sakarya, and Gregory "Lee" Schug. I have also valued the friendship of Terry "Hope Sack" Crone, Kevin "Happy" Hargaden, Faouzi Kossentini, Nancy "Nance" Lillo, Keith "Mad Dog" Matlack, and Kambiz Nayebi, as well as those in the Emmanuel Catholic Charismatic Prayer Group and the Georgia Tech Turkish Community. I would like to thank Kay Gilstrap and Stacy Schultz for their help.

I would like to thank Halûk Aydınoğlu, Wilson Chung, José Crespo, and Faouzi Kossentini for helping me proof-read my thesis. I would also like to thank Faouzi and his wife Faten for their extraordinary help in setting up for the oral defense.

I am so thankful to the Creator God for continuing to sustain me physically, emotionally, and spiritually. I am indebted beyond repayment for the love shown by Jesus Christ and the guidance given by the Holy Spirit. God gave me the strength to get through the final push to finish the thesis.

I am also thankful to God for communicating with us in many varied ways. I have tried to keep in mind that "what is seen is transitory, and what is unseen lasts forever" (II Corinthians 4:20).

Trademarks

Connection Machine is a trademark of Thinking Machines Inc.

ILS is a trademark of Signal Technology Inc.

Macintosh is a trademark of Apple, Inc.

MACSYMA is a trademark of MIT.

Maple is a trademark of Waterloo University, Canada.

Mathematica is a trademark of Wolfram Research Inc.

MATLAB is a trademark of The Math Works Inc.

MS Windows is a trademark of MicroSoft.

Monarch is a trademark of The Athena Group.

NeXT is a trademark of NeXT Computer, Inc.

N!Power is a trademark of Signal Technology Inc.

POSTSCRIPT is a trademark of Adobe Systems Inc.

Signal Processing WorkSystem is a trademark of Comdisco Systems Inc.

TEX is a trademark of the American Mathematical Society.

Unix is a trademark of AT&T.

X Windows is a trademark of MIT.

Copyrights

Contents

List of Figures

List of Tables

Summary

This thesis discusses the design and analysis by computer of algorithms composed of linear periodically time-varying (LPTV) multidimensional systems. Analysis of linear systems is based on linear transforms (e.g. z and Laplace transforms). Algorithm design rewrites component systems to reduce the implementation cost.

To support algorithm design for multidimensional systems, the thesis derives the rules for rewriting the interconnections of discrete-time LPTV multidimensional systems, a.k.a. multidimensional multirate systems, as well as develops metrics to measure implementation costs. We encode the rewrite rules, number theoretic algorithms underlying the rules, and cost metrics in a set of multidimensional signal processing packages (MDSPPs) for the computer algebra program *Mathematica*.

For algorithm analysis, the MDSPPs implement the multidimensional multisided forms of the commonly used linear transforms, and the transforms can justify their answers with natural language. Using the transforms, the MDSPPs can deduce ranges on free parameters to guarantee stability and generate input-output relationships. Engineers can use the MDSPPs to visualize signals in transform domains.

The MDSPPs represent signals as functions and systems as operators. In such an algebraic framework, the many interconnections in complex systems cannot be captured. The MDSPPs can, however, convert an algorithm for layout in the Ptolemy block diagram environment, which fits the MDSPPs into a rapid prototyping process.

Please direct any electronic correspondence concerning this research to Brian Evans at `evans@eedsp.gatech.edu`. A freely distributable version of the MDSPPs and Notebooks is available via anonymous FTP to `gauss.eedsp.gatech.edu` (IP #130.207.226.24). An estimated 10,000 people use these extensions.

CHAPTER 1

Introduction

Signal processing algorithms are often expressed as either flow graphs or algebraic formulas. As flow graphs, algorithms are decomposed into blocks with each block consisting of a set of input signals, an operator, and a set of output signals. In an algebraic framework, algorithms are decomposed into a sequence of formulas containing nested calls to operators (systems) which take functions (signals) as arguments. Operators map one or more input signals to one or more output signals. A primary drawback of an algebraic representation of algorithms is that it cannot capture the complex interconnections characteristic of large systems that are easily represented by block diagrams.

This thesis describes a software environment that takes advantage of both algebraic and block diagram representations of algorithms by

- automating the *design* and *analysis* of algebraic forms of signal processing algorithms and
- *converting* the algebraic forms of algorithms to block diagram representations for simulation and prototyping.

In the new environment, analysis of signal processing formulas is primarily based on linear transforms of multidimensional signals— the z, discrete-time Fourier, and discrete Fourier transforms of discrete-time signals, and the Laplace and Fourier transforms of continuous-time signals. Design capabilities include the interactive graphical design of two-dimensional decimation systems and the symbolic rearrangement of linear, multidimensional, periodically time-varying, discrete-time systems, i.e. multidimensional multirate systems. The conversion of algorithms to a block diagram

representation enables designers to insert developed algorithms into a layout of a complex system.

Our new environment encodes a comprehensive collection of rules and routines underlying signal processing in a set of multidimensional signal processing packages (MDSPPs) [1, 2, 3] for the *Mathematica*^{TM} [4] computer algebra program. For example, the MDSPPs can apply commutative and associative rules to switch the order of linear shift-invariant operators [5]. The MDSPPs implement many key routines such as the family of Smith form integer matrix decompositions which unlocks efficient implementations of the generalized multidimensional discrete Fourier transform when the decompositions are applied to the underlying sampling matrix [6].

In combining the rules and routines underlying signal processing into a single framework, we have been influenced by the computing environments for

- **algorithm design** especially SPLICE [7, 8, 9] and ADE [10, 11, 12],
- **symbolic mathematics** especially *Mathematica*, and
- **rapid prototyping** especially Ptolemy [13, 14, 15, 16].

SPLICE and ADE are interpretive environments for the design of one-dimensional multirate signal processing algorithms represented as formulas. These two environments can analyze properties of signals and find better implementations of algorithms. SPLICE and ADE as well as *Mathematica* and other computer algebra programs represent algorithms as formulas so they cannot capture the interconnections inherent in large complex systems. Using hierarchical block diagram representations, Ptolemy can represent and simulate complex systems as well as generate the equivalent C, DSP assembly language, and VLSI Hardware Descriptive Language (VHDL) code for complex systems. Essentially, our MDSPPs not only extend the ideas present in SPLICE and ADE to multidimensional multirate systems but also fit into a rapid prototyping process for signal processing algorithms.

1.1 Rapid Prototyping

In designing signal processing systems, signal processing theory is translated into hardware and software. That is, the mathematical principles underlying the signal processing theory are combined with knowledge of computer architectures and programming to produce a working system. The design process typically begins by translating specifications into signal processing operations. Once the component operations are identified, they are individually designed by modifying old algorithms or by developing entirely new ones. The complete system is usually simulated before it is assembled into a prototype. Table 1.1 outlines the prototyping process.

Design tools, as shown in Table 1.1, often address particular aspects of prototyping, but they rarely have the ability to transfer design information directly to other computer-aided design (CAD) tools. Hence, the prototyping process becomes fragmented because algorithms cannot be exchanged between environments. Furthermore, side information about algorithms representing lessons learned during one prototyping stage cannot be electronically transmitted to other stages. Today, many "isolated" tools are used in prototyping signal processing systems, thereby driving up prototyping costs. "For example, in a recent moving target Radar system, the signal processor accounted for over 50% of the total material cost of production" (page 10 of [17]). Simply put, the complexity of prototyping signal processing processors has outpaced the sophistication of and interaction between CAD tools [17].

1.2 Algorithm Development

In this thesis, we discuss a software environment that is useful in the algorithm development phase of prototyping. By algorithm development, we mean more than numerical simulation of algorithms, i.e. tweaking numeric parameters until the algorithm performs well enough. We also mean the symbolic reasoning about algorithms that occurs before and after simulation. Symbolic reasoning involves *analyzing* and

Development Activity	Typical Functions	Tool Availability	Typical Sources
System Engineering	System Requirements Traceability	Sparse	GEC Marconi
	Requirements Flowdown		i-Logix
	Algorithm Development		MathWorks
Design/ Upgrade	Module Physical Design & Layout	Extensive	Mentor
	VHDL Synthesis		Synopsis
	Interactive Software Development Environment		Centerline
Simulate	VHDL Simulation	Extensive	Vantage
	Module Simulation		LSI Logic
Fabricate	Pick and Place Systems, with CAD design download capability	Extensive	N. American Phillips
Integrate	Tester Interfacing	Less Extensive	Test Software Services Inc. (TSSI)
Final Test	High fidelity simulation of "outside world" for functional performance test	Extensive/ Evolving	XpertTest
	Generation of boundary scan testing at up to the system level		Hewlett Packard

Table 1.1: The Prototyping Process and Examples of Supporting Tools (adapted from Table 1 in [17]; see Appendix A of [17] for more information)

finding optimal implementations for algorithms. Analysis characterizes signals and systems in terms of their properties which tend to be symbolic relationships rather than numeric values. Optimal implementation seeks to rearrange the form of the algorithm until some cost function is minimized.

Chapter 2 reviews three environments for analyzing and finding optimal implementation of algorithms— the Signal Processing Language and Interactive Computing Environment (SPLICE), the Algorithm Design Environment (ADE), and Meta Morphology (MetaMorph) [18, 19, 20, 21]. As previously mentioned, SPLICE and ADE manipulate one-dimensional signals and (multirate) systems and attach properties to signals and systems. SPLICE and ADE can deduce signal properties such as bandwidth and extent. On the other hand, system properties are used to rearrange systems, e.g. the order of two systems in cascade can be switched if both systems are linear and shift-invariant. By applying rules that rearrange cascaded and parallel combinations of systems, Extended SPLICE (E-SPLICE) and ADE find optimal implementations of one-dimensional (multirate) systems by minimizing a simple cost function. MetaMorph manipulates morphological signals (sets) and systems (nonlinear set operators) in a similar way. In order to fit into a rapid prototyping process, however, an environment should be able to generate equivalent code for algorithms that would be suitable for incorporation by tools at the next level of prototyping. Unfortunately, ADE, SPLICE, and MetaMorph cannot generate code for algorithms.

1.3 The Multidimensional Signal Processing Packages (MDSPPs)

The goal in creating our new algorithm development environment was to complement the abilities of the SPLICE, ADE, and MetaMorph environments. As previously mentioned, our new environment is a set of multidimensional signal processing packages (MDSPPs) for *Mathematica* that analyze and find optimal implementations of

linear, *multidimensional*, multirate signal processing algorithms. Unlike SPLICE and ADE, which only run on Symbolics machines, the MDSPPs are available for many computing platforms because they run wherever *Mathematica* does. Unlike SPLICE, ADE, and MetaMorph, the MDSPPs fit into a rapid prototyping framework because it generates block diagram descriptions for algorithms which can be inserted into the layout of a complex system in Ptolemy. We chose Ptolemy because it can represent and simulate complex systems, as well as generate the equivalent C code, DSP assembly programs, and VHDL descriptions for complex systems.

Although SPLICE, ADE, and MetaMorph have been written in `Lisp`, we have chosen to start at a higher level by using the symbolic mathematics program *Mathematica*. *Mathematica* offers many benefits. *Mathematica* is proficient at partial fraction decomposition, power series expansions, symbolic integration, and infinite summations, which are fundamental operations in convolution and linear transforms. *Mathematica* also provides a mechanism for expressing and applying rules. Because of the various plotting and graphics formats in *Mathematica*, users of the MDSPPs can visualize (in different domains) signals generated by one-dimensional and two-dimensional (multirate) systems. *Mathematica* requires any routine accessible by the user to have usage information attached to it, so all new functions come with help information. *Mathematica* also supports the playing of sound and animation. On several platforms, *Mathematica* offers a multimedia Notebook user interface [22] that organizes formatted text, sound, graphics, animation, and *Mathematica* code in a tree structure. Using the Notebook interface, we have written a user's guide, several tutorials, and on-line help in the form of interactive electronic documents [3, 23, 24, 25, 26].

1.4 Layout of the Thesis

As previously mentioned, Chapter 2 discusses the SPLICE, ADE, and MetaMorph algorithm design environments and evaluates *Mathematica* as a platform upon which to build an algorithm design environment. Chapter 3 unifies the theory underlying the operation of and interaction between multidimensional multirate structures. Based on this new theory, we develop the rules that generalize the interaction between up/downsamplers and linear shift-invariant operators to multiple dimensions in such a way as to be easily implementable by computer. Chapter 4 introduces the representation of signals and systems in the MDSPPs.

The subsequent chapters discuss the algorithm development capabilities of the MDSPPs. The MDSPPs

1. analyze linear multidimensional multirate structures (Chapter 5),
2. find optimal implementations for algorithms involving these structures (Chapter 6),
3. generate equivalent TEX^{TM} and Ptolemy code for these structures (Chapter 7), and
4. assist in the interactive design of two-dimensional rational decimation systems (Chapter 8).

The ability to generate complete Ptolemy simulations for algorithms links the MDSPPs to the rapid prototyping process.

The higher-level functions of the MDSPPs feature the ability to justify their answers in the form of textual and/or graphical dialogue. For example, the linear transform routines can show the intermediate hand calculations involved in taking z, Laplace, and other transforms, whereas the convolution routines can illustrate by animation the flip-and-slide approach to convolving two piecewise functions. These

dialoguing capabilities have helped students learn the theory underlying signal processing [3, 23, 24, 25, 26]. The Notebooks accompanying the MDSPPs guide a student's exploration of signal processing theory. Chapter 9 assesses the impact of the our new environment on engineering education.

Chapter 10 highlights the contributions of the thesis and summarizes areas of future research. Appendix A gives two examples of Ptolemy code generation for a two-channel one-dimensional non-uniform filter bank. Appendix B is a glossary of terms.

1.5 Scope of the Thesis

This thesis treats several diverse topics that may not appeal to all readers. Readers with a signal processing background would probably be the most interested in Chapters 3, 5, and 8 as they characterize, analyze, and design multidimensional multirate systems, respectively. They would also be interested in Chapter 4 as it introduces signal processing with formulas by computer, which is a review of the algebraic representation of signals and systems as functions and operators, respectively.

Mathematicians would probably be interested in the same chapters (namely Chapters 3, 4, and 8) but for different reasons. Chapter 3 applies lattice theory and integer matrix algebra to describe the regular insertion and deletion of points on a uniform grid known as upsampling and downsampling, respectively. Chapter 4 derives an algorithm that performs convolution with functions described in a piecewise manner— each interval consists of a formula that defines the function on that interval and a pair of endpoints that can be symbolic (e.g. infinite). Chapter 8 presents three design examples that require both exact precision and floating point arithmetic in the calculation of free parameters. Although not discussed in this thesis, a mathematician may be interested in the use of a computer algebra environment to derive, encode, and test new number theoretic algorithms for integer matrix decompositions which

we present in [27].

On the other hand, computer scientists would most likely be interested in Chapters 6 and 7. Chapter 6 discusses algorithm rearrangement using forward-chained context-free rules. Chapter 7 explores automatic code generation (program synthesis) for formulas using bottom-up parsing of the algorithm represented as a tree of operations and a hash table to prevent the generation of redundant operations.

Educators would most likely be interested in Chapter 9 because it explores the use of computer algebra systems to teach linear systems (transform) theory. We have written *Mathematica* routines so that they can show students how to perform the calculations by hand. Using *Mathematica*'s Notebook interface, we have developed interactive tutorials to enable students to explore new topics. Other Notebooks help students test themselves.

In essence, this thesis weaves together concepts from many different fields. These concepts were necessary to make an algorithm development environment that is simultaneously useful to designers implementing signal processing algorithms, students learning linear systems theory, and researchers exploring new signal processing theory. Because of the multidisciplinary nature of this thesis, we provide a glossary of terms in Appendix B.

CHAPTER 2

Background

Many environments have been developed for the automatic simulation, interpretation, and design of algorithms, as shown in Figure 2.1. In testing algorithms, engineers simulate them by passing a variety of numerical signals into the algorithm until the output signals exhibit the desired characteristics. For some domains, signal interpretation can be performed on numeric signals to produce symbols as the output (e.g. words in speech recognition). Sometimes, engineers need assistance deriving new algorithms or rearranging old algorithms to achieve a better performance (maybe fewer multiplications and additions per output sample). In this case, the engineer begins with a symbolic (possibly mathematical) description of an algorithm and then tries to manipulate it into another symbolic description having a more desirable form.

Many of the computer tools available for simulation, interpretation, and design of signal processing algorithms originated in the work of Kopec [28, 29, 30]. Kopec developed a general simulation environment for causal numeric signals, but he set

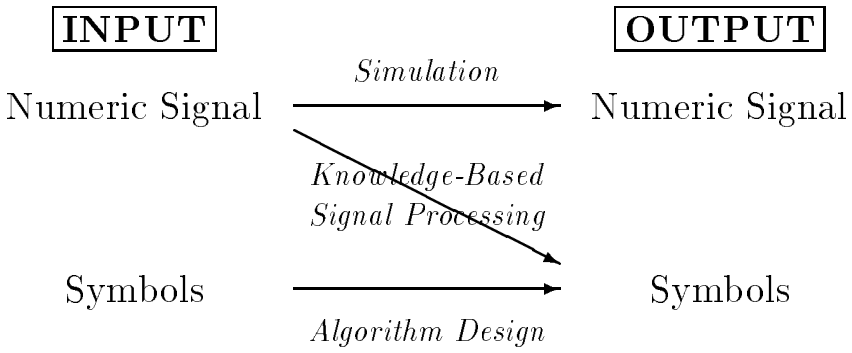


Figure 2.1: Categories of Existing Algorithm Support Tools

the stage for future environments by introducing an object-oriented representation of signals. Signals, as objects, contain slots (data) and methods (procedures). Slots include the signal's history and previously computed values. Methods include those describing how to compute signal values. Kopec's interpretive Signal Representation Language (SRL) influenced the development of environments for the *simulation*, *knowledge-based interpretation*, and *design* of signal processing algorithms, as shown in Figure 2.2. All three types of environments, especially the latter two, are discussed in [31]. The first half of [31] discusses algorithm design environments in detail, whereas this chapter only provides an overview of them.

The first algorithm design environment to follow Kopec's work is the Signal Processing Language and Interpretive Computing Environment (SPLICE) by Myers [7, 8, 9]. Myers extended Kopec's representation in the direction of "abstract" signals. Abstract signals are defined in terms of their properties rather than in terms of their sample values. Myers also introduced abstract systems. Combining abstract signals and systems led to the ability to simplify and rearrange one-dimensional (multirate) signal processing algorithms. By enumerating equivalent implementations, Extended SPLICE (E-SPLICE) can choose implementations that are "optimal" according to a simple cost function. Covell [10, 11, 12] improved upon E-SPLICE to create the Algorithm Design Environment (ADE). ADE applies heuristics to reduce the number of alternative implementations generated, and it uses a more complicated cost function. Richardson [18, 19, 20, 21] applied the same concepts to morphological signals (sets) and systems (set operators) in his MetaMorph environment.

Section 2.1 describes Kopec's work. Section 2.2 discusses the SPLICE, ADE, and MetaMorph algorithm design environments. Although these environments have been written in `Lisp`, we have chosen to write ours in the *Mathematica* symbolic mathematics program. Section 2.3 evaluates *Mathematica* as a platform for algorithm design.

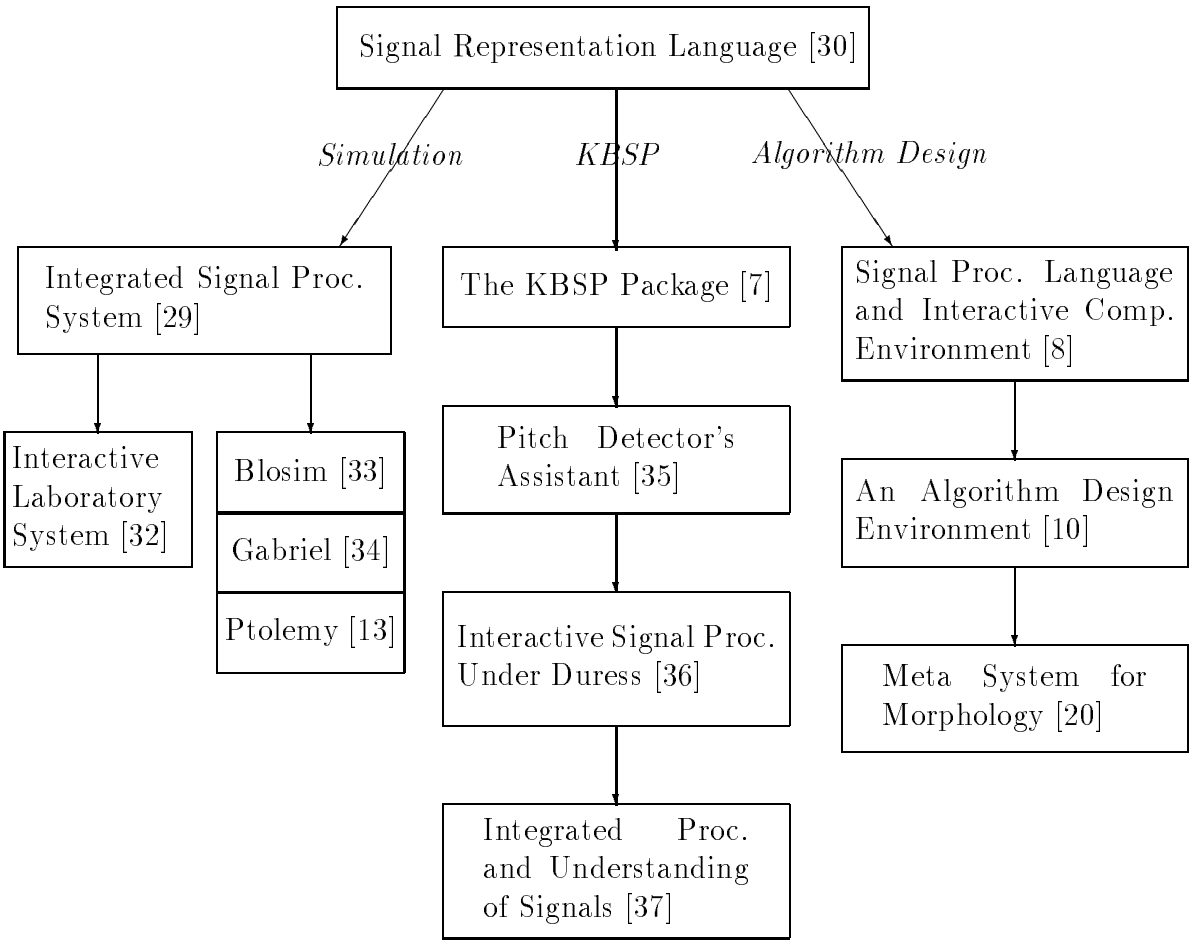


Figure 2.2: Environments Influenced by the Signal Representation Language

2.1 Representing Signals as Objects

In his thesis, Kopec identifies different models for computing and obtaining signal values [28]. As functions, signals can be computed point-by-point (and for efficiency, computed values are stored in a cache for future use). Finite-length signals can be captured in an array. Data streams [38] (e.g. modems) and state-space models [39] (e.g. Kalman filters) form two important classes of infinite-duration signals. These four classes provide a way of *abstracting* signal computations. That is, a programmer could ask for values of a signal without having to know how they were computed.

Hiding the method of computation is a principle of object-oriented programming. As an object, a signal has data (slots) and procedures (methods) attached to it. Under this paradigm, the description of a signal becomes more than a domain and range of values— it can now contain information about its parameters, history, computational model, and so forth. This extended representation of a signal is implemented in Kopec's Signal Representation Language (SRL) [30]. SRL hierarchically classifies signals and then instantiates signals as needed. Once signals are created, their properties remain fixed. This *immutability* of signals is the result of the closure model for signals [31] and is necessary during the generation of alternate implementations.

SRL abstracts the representation of signals so as to aid in their computation but restricts signals to be one-dimensional finite-extent causal sequences beginning at time index 0. Systems are implemented as `Lisp` routines that map one numeric array into another. SRL can simulate algorithms that use these representations of signals and systems, but it cannot symbolically reason about signal properties.

Because interaction with SRL was limited to a command line interface using a `Lisp` syntax, Kopec developed the Integrated Signal Processing (ISP) System [29] which provided a graphical user interface to SRL. The ideas behind SRL and ISP were recently implemented in the program *N!Power*^{TM} [40].

Signal Property	Meaning
`Bandwidth`	non-zero extent of frequency domain
`Center-of-Symmetry`	center of symmetry in the time domain
`End`	end of non-zero extent in the time domain
`End-BW`	end of bandwidth
`Period`	period
`Real-or-Complex`	value is `REAL` or `COMPLEX`
`Start`	start of non-zero extent in the time domain
`Start-BW`	start of bandwidth
`Support`	non-zero extent in the time domain
`Symmetry`	multi-valued parameter describing time-domain symmetry; possible values are: `SYMMETRIC`, `ANTISYMMETRIC`, `CONJUGATE-SYMMETRIC`, and `CONJUGATE-ANTISYMMETRIC`

Table 2.1: Signal Properties in SPLICE, E-SPLICE and ADE

2.2 Algorithm Design Environments

In his Signal Processing Language and Interactive Computing Environment (SPLICE), Myers further generalizes the representation of one-dimensional discrete-time signals begun by Kopec. SPLICE allows signals to be infinite in extent and to have arbitrary starting and ending points. Besides keeping track of time-domain characteristics of a signal, SPLICE captures information about the signal's bandwidth and other signal properties (Table 2.1).

In an extended version of SPLICE (called E-SPLICE), Myers introduces "abstract" signals. These are signals that do not have signal values associated with them. Instead, they are defined completely by their properties. This notion allows E-SPLICE to manipulate signals at a higher level of abstraction.

E-SPLICE also applies abstraction to systems by assigning properties to them (see Table 2.2). Unlike signals, however, systems do not have to be instantiated because their properties do not change with different values of parameters. For example,

System Property	Meaning
`ASSOCIATIVE`	can change grouping of inputs
`ADDITIVE`	distributes over addition
`COMMUTATIVE`	can change order of inputs
`HOMOGENEOUS`	scaled input gives scaled output
`LINEAR`	additive and homogeneous
`MEMORYLESS`	output does not depend on previous inputs or outputs; if a single-input system, then `SHIFTINVARIANT`
`SHIFTINVARIANT`	shifted input gives shifted output

Table 2.2: System Properties in E-SPLICE and ADE

shifting by L with respect to n is linear and shift-invariant regardless of the value of L.

Based on the properties of systems, Myers developed rules to simplify and rearrange parts of signal processing expressions. Using these rewrite rules, E-SPLICE can generate equivalent forms of algorithms and compare their relative costs by measuring the number of additions and multiplications required. Unfortunately, E-SPLICE blindly applies its rewrite rules. As a consequence, a combinatoric explosion in computer memory and computation time can result for complicated expressions. Furthermore, the equivalent forms may loose the regularity (parallelism) inherent in the original form.

E-SPLICE set the stage for Covell's Algorithm Design Environment (ADE) [10]. ADE is a recasting of E-SPLICE from Symbolics `Zetalisp` into Symbolics Common `Lisp`. Unlike E-SPLICE, ADE allows new abstract systems to be defined. New systems are defined in terms of their properties. Since rules are already associated with the properties, only rules not based on properties have to be added to handle special cases. Describing systems in terms of their properties prevents a combinatoric explosion in the number of rules required as new systems are added.

Compared to E-SPLICE, ADE not only encodes new simplification and rewrite

Year	Name	Field of Knowledge	Simulation	Signal Properties	System Properties	Algorithm Design
1985	SRL	1-D DSP	X	X		
1986	E-SPLICE	1-D DSP	X	X	X	X
1989	ADE	1-D DSP	X	X	X	X
1991	MetaMorph	Morphology	X	X	X	X

Table 2.3: Overview of Algorithm Design Environments

rules, but the engine that applies the rewrite rules is more sophisticated. The engine prunes the number of rewrite rules to apply to an algorithm. For example, the engine would apply a rule to a set of parallel branches in an algorithm only if the parallelism (regularity) in the branches would be maintained. ADE also uses a more complicated cost function. Instead of being able to count only the number of multiplications and additions, ADE can also count the amount of memory needed and express inequality relationships between individual cost measures, e.g. "1 complex multiply is always more expensive than 3 real multiplies" [41]. Like SRL and SPLICE, ADE can also simulate algorithms.

Kopec, Myers, and Covell have focused their work on linear one-dimensional signals and systems. Richardson [18, 19, 20, 21] has applied their ideas to the design, analysis, and simulation of morphological algorithms. Morphological signals are sets of points, and morphological systems (non-linearly) map one set of points to another. Richardson identified signal and system properties based on set theory and encoded a comprehensive collection of simplification and rearrangement rules in his MetaMorph environment. To aid in the finding of optimal implementations for morphological algorithms, Richardson developed a new cost metric that takes into account the number of maximum, minimum, union, and intersection operations as well as the number of additions and multiplications. MetaMorph intelligently applies its rewrite rules.

2.3 *Mathematica* Programming for KBSP

SPLICE, ADE, and MetaMorph are implemented in `Lisp`, whereas we have decided to forge an algorithm design environment out of *Mathematica*. *Mathematica* [4], released in 1988, is a general mathematics program like Maple^{TM} [42] and MACSYMA^{TM} [43]. All three programs are standardized, documented, portable, programmable environments. Because they embody extensive knowledge about mathematical structures and operations, their forte is the manipulation of formulas. For example, they compute $7x + 3x$ as $10x$. They are proficient at factoring polynomials and performing partial fraction expansions as well as differentiating and integrating expressions. Differentiation and integration are examples of symbolic operations that need to know which symbols to treat as variables. For example, the expression $C\tan(x)$ contains two symbols, C and x. Integrating with respect to x yields $-C\log(\cos(x))$, but integrating with respect to C yields $C^2\tan(x)/2$. Symbolic mathematics programs can also represent and compute sequences from recursive formulas.

Since signals can be represented as functions and systems as mathematical operators that map signals to other signals, many signal processing algorithms can be expressed in algebraic form as a sequence of formulas. Formulas can compute signal values point by point or by array operations. In general, symbolic mathematics environments do not support streams or state-space computational models [28] although they can be programmed to do so. All three mathematics programs can implement one-to-one transformations as recursive function calls so that well-defined operations like linear transforms can be conveniently defined. Alternately, *Mathematica* and MACSYMA can implement one-to-one transformations by applying a list of conditional rules, whereas Maple lacks even the ability to express conditional rules because it does not support pattern matching. Unfortunately, *Mathematica* and MACSYMA do not have built-in mechanisms to apply rules for transformations that are not one-to-one (e.g. searching for optimal implementations of an algorithm).

2.3.1 *Mathematica*

We have chosen *Mathematica* primarily because of its Notebook front end. The Notebook front end provides pull-down menus, screen- and mouse-oriented editing of previous commands, and a POSTSCRIPT^{TM} driver for all graphics. Graphics, animation, sound, formatted text, and *Mathematica* code can co-exist in the same Notebook. Notebooks typically arrange information as a hierarchical grouping of cells, so a user can quickly and randomly access information. A user can also choose to interact with these *active documents* by modifying the *Mathematica* code to try new examples [22]. Although the Notebook front end is a multimedia platform that groups ideas in a tree structure, it is not a true hypertext platform as individual ideas are not interconnected in a web structure [44]. As a multimedia platform, however, the off-line documentation can be identical to the on-line documentation. The Notebook interface exists for several windowing systems (e.g. Macintosh^{TM}, MicroSoft^{TM}, NeXT^{TM}, and X^{TM}). Without the Notebook facility, the front end defaults to a simple teletype (tty) terminal interface. On Unix^{TM} machines, however, the Emacs editor can be configured to run as a front end to the *Mathematica* kernel.

Although the user interface is machine-dependent, the back-end *kernel* is the machine-independent computational engine. The kernel provides a powerful pattern matcher, conditional rule formats, arbitrary precision arithmetic, and a programming language. This programming language organizes code into packages (modules) much as `Lisp` does. Embedded in the kernel are packages that perform integration, differentiation, matrix manipulation, and linear programming. The kernel provides powerful symbolic operations such as differentiation and integration as well as hundreds of computational functions such as the multidimensional numerical discrete Fourier transform (sampled on a rectangular grid). If initialized properly, *Mathematica* will automatically load new functions as they are invoked by the user [4, 45].

Making the transition to *Mathematica* from a high-level programming language is not difficult. Internally, *Mathematica* maintains expressions in prefix form as Lisp

mathematical operations	`+`	`-`	`*`	`/`	`++`	`--`
	`=`	`+=`	`-=`	`*=`	`/=`	
relational operators	`<`	`<=`	`>`	`>=`	`==`	`!=`
bit operations	`|`	`&`				
logical operators	`&&`	`||`				

Table 2.4: *Mathematica* Operators That Mimic C Syntax

does. *Mathematica* functions have the form

$$function\ [\ argument_1,\ argument_2,\ \ldots\ ]$$

Mathematica data structures have the same format:

$$dataType\ [\ field_1,\ field_2,\ \ldots\ ]$$

Mathematica's pre-processor enables the user to express function calls in infix or post-fix format. The pre-processor mimics most of the standard C operators, which are either in infix or post-fix notation, as shown in Table 2.4. Many of *Mathematica*'s list-processing primitives such as `First` and `Rest` are borrowed from Lisp, as well as the *Mathematica* primitives `Print`, `Apply`, `Map`, and `MapAll`. A user can customize the pre-processor so that it can recognize a syntax that is more familiar. The novice can learn more about *Mathematica* from on-line documentation (written as *Mathematica* Notebooks) or from the more than 20 books written on *Mathematica* (we highly recommend [46]). The help facility built into the kernel, abbreviated as `?`, is more useful when users already know the name(s) of the routine(s) they need.

2.3.2 *Mathematica* As a Platform for Algorithm Design

Mathematica presents an attractive platform upon which to build an environment for algorithm design. As previously mentioned, the kernel provides ways to write programs and encode rules [45]. In fact, many different programming paradigms [38] are offered: procedural, functional, object-oriented, constraint-based, and rule-based.

The `Block` and `Module` constructs are useful for writing procedures, and the `Function` primitive generates the equivalent of function pointers [47]. *Mathematica* does not implement object-oriented programming *per se* but instead allows the programmer to attach data and code to a data type. When using such an data-directed approach, a programmer can write code that is incrementally extensible so that the addition of a new data type (tag) does not require modification of existing code. Constraint propagation, implemented by the `Solve` command, only applies for equality constraints. Rules in *Mathematica* are specified using an *if...then* form. When the conditional part of the rule is satisfied, its consequence is evaluated. The consequence expression can reference patterns matched during the evaluation of the conditional clause. This rule framework only supports reasoning with certainty.

These programming styles offer two direct advantages over high-level languages for implementing signal processing operations. First, as a consequence of support of the functional programming paradigm, cascaded systems (operators) become nested calls to *Mathematica* objects. Second, *Mathematica* provides several ways to encode rules and apply them to expressions. One way is to attach rules to object heads (function names). *Mathematica* will fire such rules whenever an expression contains that function. For example, the relationship

$$\Re e\{x + y\} = \Re e\{x\} + \Re e\{y\}$$

can be rendered in *Mathematica* by either attaching the rule to the `Re` or `Plus` built-in primitives:

```
Re/:    Re[x_ + y_] := Re[x] + Re[y]
Plus/:  Re[x_ + y_] := Re[x] + Re[y]
```

The `:=` defines a procedure in *Mathematica*, but unlike many high-level languages, several definitions can be attached to an object to cover different combinations of arguments. The underscore character indicates a pattern to be matched (`x_` is analogous to `?x` in Lisp-based pattern matchers [48]). If the first definition above is evaluated,

Mathematica's read-evaluate loop will check this rule every time the real part of any quantity is sought, even though the rule does not apply. Therefore, the first but not the second definition will slow down computations involving the `Re` function such as taking the absolute value of a complex number because *Mathematica*'s `Abs[x]` function is implemented as `Sqrt[ Re[x]^2 + Im[x]^2 ]`.

For any complicated operation, it is desirable for a knowledge-based environment to justify its answers. Students may want to learn how to perform the operation by hand, whereas designers and researchers may want to verify the approach taken by the knowledge base. When the knowledge base evaluates a recursive operation (such as one-to-one transformations) as recursive function calls, the order of operations may follow a pre-order traversal of the tree form of the current expression. For example, if the Fibonacci sequence were defined as

$$f_n = f_{n-1} + f_{n-2} \qquad f_0 = 0 \qquad f_1 = 1$$

then tracing calls to f for f_8 would first show $f_7 + f_6$, but then it would show the computations involved in computing f_7. After f_7 had been computed, then the computations for f_6 would be shown. In this case, displaying the intermediate calculations would only show the transformation of nested sub-expressions as they are decomposed (i.e. a depth-first traversal of the tree of operations to compute). Such a dialogue would not appear natural or be easy to follow because all of the terms involved in the computation would not be shown at each intermediate step.

For the purposes of displaying natural dialogue, a more desirable way to encode recursive operations is to collect related rules in a list (called a rule base) and apply them recursively (e.g. using `ReplaceRepeated`). Since the rules are not attached to any function name (head), this approach avoids slowing down unrelated computations, as was the case when we attached rules to the `Re` primitive. Another benefit is that a programmer can control how the rules are applied. For one-to-one transformations, the programmer can take the expression to be transformed, apply the first valid rule, and display the intermediate result. By repeating this method until the

transform is computed, the resulting dialogue would mimic how a human would take the transform. This method also allows the user to decide when to fire the rules. Our multidimensional signal processing packages (MDSPPs) implement transforms using this rule-based approach.

Being a symbolic mathematics environment, *Mathematica* does more than apply rules. For example, two primitives— partial fraction decomposition (`Apart`) and power series expansion (`Series`)— are critical for taking inverse transforms. The ability to factor, expand, and rearrange polynomials plays another key role in computing symbolic transforms. A programmer can combine these mathematical operations with the rule-based approach to encode powerful routines to perform the z-transform and other one-to-one transformations. The user can extend the routines by simply adding transform pairs as rules.

Besides implementing a wide variety of programming styles, this environment provides other advantages for algorithm design:

1. dynamic data typing,

2. diverse graphics capabilities, and

3. code generation.

Dynamic data typing allows valueless symbols and permits variables to assume any data type, although *Mathematica* 1.0 generally assumes that unassigned symbols represent a real-valued quantity whereas *Mathematica* 2.0 generally assumes that unassigned symbols represent a complex-valued quantity. The kernel can graph one-dimensional functions and parametric relationships. It also supports scatter, density, and contour plotting as well as three-dimensional graphics. On general purpose computers, *Mathematica*'s ability to generate the equivalent Fortran, C, and TEX code for mathematical formulas (by default) and signal processing expressions (by extension) becomes useful, because the new code can be spliced into existing programs and documents.

The *Mathematica* kernel does come with some disadvantages for algorithm design. First, the level of abstraction means that numerical computations are slower, although Version 2.0 introduces the notion of compiled functions to speed this up. Second, *Mathematica* can only chain rules in one direction— it is not an inferencing engine. Third, the rule rewriting mechanism applies a list of production rules in sequential order to transform one expression into another. That is, *Mathematica* does not generate a tree of all possible new expressions, and therefore, it does not provide heuristic techniques to search through a solution space. Finally, common signal processing operators are, for the most part, not available in *Mathematica*. The multidimensional signal processing packages overcome the last two deficiencies.

2.4 Summary

This chapter summarizes previous algorithm design environments which have their origins in Kopec's work [28, 29, 30]. Kopec's key contribution was the abstraction of signals as self-contained objects. In subsequent work, Myers applied abstraction to systems and enhanced the abstraction for signals [7, 8, 9]. Combining abstract signals and systems lead to the ability to simplify and rearrange one-dimensional (multirate) signal processing algorithms and, ultimately, to find optimal implementations of algorithms based on a simple cost function. Inspired by Myers' work, Covell developed efficient inferencing strategies to apply rearrangement rules in the finding of optimal implementations with the option of maintaining parallelism in the algorithm [10, 11, 12]. Richardson has applied the same concepts to morphological signals and systems in his MetaMorph environment [18, 19, 20, 21].

In developing their algorithm design environments, Myers, Covell, and Richardson had to extend the Lisp language to implement pattern matching, inferencing strategies, mathematical abilities, and object-oriented programming. Because Myers and Covell encoded SPLICE and ADE (respectively) in Symbolics dialects of Lisp,

SPLICE and ADE are not available for general use. SPLICE, ADE, and META-MORPH do not support graphics and offer only a command line interface.

Instead of beginning with Lisp, we have chosen to start at a higher level by using the symbolic algebra program *Mathematica*. The *Mathematica* programming language is essentially Lisp without garbage collection but with pattern matching, forward chaining of rules, and a simple implementation of objects built in. The *Mathematica* kernel adds hundreds of mathematical operations and many plotting routines. Although the *Mathematica* kernel does not possess much knowledge about signals and systems, we have programmed it to do so. The result is a set of multidimensional signal processing packages (MDSPPs) for *Mathematica* which are described in Chapters 4–8.

Mathematica is a widely available standardized documented portable environment. On some platforms, *Mathematica* offers a sophisticated multimedia Notebook user interface. We have written several Notebooks to accompany the MDSPPs so that the environment is self-documenting. Chapter 9 discusses the signal processing Notebooks.

CHAPTER 3

Multidimensional Multirate Signal Processing

The ultimate goal of the new multidimensional signal processing packages (MDSPPs) for *Mathematica* is to aid in the development of multidimensional multirate signal processing algorithms. At the heart of these algorithms lies the linear periodically time-varying discrete-time operations of upsampling and downsampling. In one dimension, upsampling and downsampling as well as their interconnections have been thoroughly researched [49, 50].

The primary purpose of this chapter is to generalize the rules for the one-dimensional multirate structures compiled in [49] to multiple dimensions. We also develop the rules for rewriting an upsampler and downsampler in cascade and an upsampler, shifter, and downsampler in cascade. In the case of the up/downsampler cascade, we analyze the algorithms underlying both the time- and frequency-domain conditions for commutativity. For completeness, we include rules reported by other authors such as polyphase implementations of multidimensional rational decimation systems (rate changers) [51, 52]. This chapter derives the algorithms underlying these rules in such a way as to be easily implementable by computer. We have encoded the rules and the underlying algorithms in the MDSPPs.

In one dimension, many of the multirate rules depend on the commutativity property of multiplication over the semigroup of non-zero integers. Because of commutativity, integers can be factored into a product of primes. In multiple dimensions, resampling is characterized by the semigroup of non-singular square integer matrices called *resampling matrices*. The product of resampling matrices rarely commutes, and

as a consequence, factoring must be generalized using left and right matrix factors. The left and right matrix factors, however, are the same for the subset of diagonal resampling matrices. Diagonal resampling matrices correspond to processing multidimensional data separately, one dimension at a time. In simplifying and rearranging cascades of upsampling and downsampling operations, it is essential to transform multidimensional multirate operators into a separable form.

In this chapter, we address several fundamental issues associated with the design and implementation of multidimensional multirate systems. Section 3.1 reviews pertinent background material including concepts from lattice theory and matrix theory (especially the Smith Form decomposition and the least common right multiple). These mathematical disciplines are needed for a rigorous theory of multidimensional resampling operations. Section 3.2 investigates the one-dimensional rules involving cascades of upsamplers and downsamplers that can be generalized to multiple dimensions by using only matrix multiplication and inversion. In Section 3.3, we apply the Smith Form decomposition of a resampling matrix to several problems: computing coset vectors, finding greatest common sublattices, and simplifying up/downsampler cascades. Section 3.4 discusses two new sets of equivalent conditions for the commutativity of an up/downsampler cascade. Section 3.5 defines other new rules and collects them all together to form a symbolic algebra for multidimensional multirate signal processing which we have encoded as a part of the multidimensional signal processing packages. We summarize our findings in Section 3.6.

3.1 Lattice and Matrix Theory

In this section, we present a summary of relevant concepts from the theory of lattices and the theory of matrices. Both are necessary for a rigorous understanding of uniformly sampled (or resampled) signals.

Multidimensional data can be sampled on rectangular, hexagonal, polar, and

other grids [53, 54, 55]. In this chapter, we only consider uniform sampling. Two common uniform sampling grids for image processing are rectangular and hexagonal. Each point on a uniform sampling grid is defined by an integer vector from the origin to the point. We will refer to the collection of all such integer vectors as $\mathcal{R}_I$. $\mathcal{R}_I$ is an example of a *lattice*, i.e. a regular arrangement of points in a space. Lattices, sublattices, resampling matrices, and other related topics are defined in Section 3.1.1. Section 3.1.2 defines multidimensional upsampling and downsampling. The remaining subsections discuss decomposing and factoring resampling matrices and the relationship of resampling matrices to lattices.

3.1.1 Definitions

The results in this chapter rely on the mathematical concepts of lattices, sublattices, cosets, and coset vectors as they relate to resampling matrices. A lattice is a sampling grid (rectangular, hexagonal, etc.), a sublattice is a subset of a lattice obtained by an integral linear mapping of a lattice, and cosets represent different ways to shift the sampling grid. A brief discussion of these ideas follows. For a more complete discussion of the geometry of numbers, the reader is referred to [6, 56, 57, 58].

Lattice: A *lattice* is a discrete set of vectors (points, n-tuples) in Euclidean n-dimensional space $\mathbf{R}^n$ that forms a group under ordinary vector addition (i.e., given two points in a lattice, their Euclidean difference and sum are also in the lattice). A lattice, $\mathcal{L}$, contains an infinite number of points

$$\mathcal{L} = \{\mathbf{n} \mid \mathbf{n} = u_1\mathbf{a}_1 + \ldots + u_n\mathbf{a}_n\} \tag{3.1}$$

where u_i is an integer, and the vectors $\{\mathbf{a}_i\}$ are linearly independent real vectors in an n-dimensional space. A lattice can be characterized by a sampling matrix whose columns are the $\{\mathbf{a}_i\}$ vectors. $\mathcal{R}_I$ will denote the *integer* lattice.

Resampling Matrix: A *resampling matrix* is a non-singular square integer matrix that maps $\mathcal{R}_I$ into itself.

Sublattice: A *sublattice* is a lattice which is contained within a larger lattice. In this chapter, we will only consider sublattices of the integer lattice $\mathcal{R}_I$. A sublattice associated with the resampling matrix S is defined as the range of S:

$$\text{sublattice}(S) \equiv \{S\,\mathbf{n} \mid \mathbf{n} \in \mathcal{R}_I\}$$

Common Sublattice: Given three lattices $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$, if $\mathcal{A}$ is a sublattice of $\mathcal{B}$ and $\mathcal{A}$ is also a sublattice of $\mathcal{C}$, then $\mathcal{A}$ is called a *common sublattice* of $\mathcal{B}$ and $\mathcal{C}$.

Greatest Common Sublattice: The *greatest common sublattice* is the union of all common sublattices.

Cosets and Coset Vectors: A *coset* is obtained by simply shifting a sublattice by an integer vector, $\mathbf{k}$. Any vector from the origin to a point on a coset is a *coset vector*. There are an infinite number of coset vectors.

Distinct Coset Vectors: The *distinct coset vectors* associated with the resampling matrix S are the $|\det S|$ integer vectors that lie within the fundamental parallelepiped (FPD) of S, denoted by

$$\text{FPD}(S) = \{S\mathbf{x} \in \mathcal{R}_I \,|\, 0 \le x_i < 1\}$$

where x_i is the ith element of $\mathbf{x}$. The distinct coset vectors provide a convenient way to enumerate a block of samples for resampling. For example, in one-dimensional downsampling (upsampling) by a factor of N, the distinct coset vectors are the indices $0 \ldots N-1$ of each block of input (output) samples being deleted (inserted).

The following Theorem and Corollary will be helpful in proving a later result. The proof is omitted here because it can be found in a variety of texts and papers on lattice theory [6, 57].

Theorem 1: The sublattice associated with a resampling matrix A contains the sublattice associated with the resampling matrix B if and only if $B = AC$ where C is also a resampling matrix.

Corollary 1.1: The sublattice associated with a resampling matrix A is equivalent to the sublattice associated with the resampling matrix B if and only if $B = AQ$ where Q is a resampling matrix such that $|\det Q| = 1$.

Corollary 1.1 is true because multiplying on the right by Q such that $|\det Q| = 1$ does not alter the coset vectors. However, multiplying on the left by such a matrix shuffles the coset vectors which in turn alters the shape of the sublattice.

3.1.2 Uniform Resampling of Lattices

The fundamental building blocks of multirate systems are the dual operations of upsampling and downsampling.

Upsampling

Definition: An *upsampler* is characterized by a resampling matrix L called the upsampling matrix. The upsampling operation with input $x[\mathbf{n}]$ and output $x_{\mathrm{u}}[\mathbf{n}]$ is defined as

$$x_{\mathrm{u}}[\mathbf{n}] = \begin{cases} x[L^{-1}\mathbf{n}] & \text{if } L^{-1}\mathbf{n} \in \mathcal{R}_I \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

and is denoted by $\uparrow L$ or $\uparrow_{\mathbf{n}} L$ or $\uparrow_{L,\mathbf{n}}$.

An upsampler maps the input signal, $x[\mathbf{n}]$, defined on $\mathcal{R}_I$, to the signal $x_{\mathrm{u}}[\mathbf{n}]$ which is also defined on $\mathcal{R}_I$. The values of $x[\mathbf{n}]$ are mapped to locations in $x_{\mathrm{u}}[\mathbf{n}]$ which are on the sublattice(L). All the values of samples of $x_{\mathrm{u}}[\mathbf{n}]$ not on the sublattice(L) are set to zero. The multidimensional Fourier transform (which is defined in [53]) relationship for an upsampler is [53, 59]

$$X_{\mathrm{u}}(\omega) = X(L^T\omega) \tag{3.3}$$

where ω is a vector of the discrete-time frequency variables. In multiple dimensions, a discrete-time frequency response $X(\omega)$ is periodic with period 2π in each discrete-time frequency variable:

$$X(\omega) \;=\; X(\omega \;+\; 2\pi\, N\, \mathbf{r}) \;\; \forall\, \mathbf{r}$$

where $\mathbf{r}$ is an integer vector and N is an integer periodicity matrix. The upsampler maps the periodicity by L^T, so if the upsampling matrix in non-rectangular, then the periodicity of the output frequency domain would be non-separable. The sampling density decreases by a factor of $|\det L|$, the upsampling factor.

Downsampling

> **Definition:** A *downsampler* is characterized by a resampling matrix M called the downsampling matrix. The downsampling operation, with input $x[\mathbf{n}]$ and output $x_d[\mathbf{n}]$, is defined as
>
> $$x_d[\mathbf{n}] = x[M\mathbf{n}] \tag{3.4}$$
>
> and is denoted by $\downarrow M$ or $\downarrow_{\mathbf{n}} M$ or $\downarrow_{M,\mathbf{n}}$.

A downsampler maps the input signal $x[\mathbf{n}]$ to the output signal $x_d[\mathbf{n}]$ by discarding samples of $x[\mathbf{n}]$ that do not lie on sublattice(M) and then compacts sublattice(M) onto $\mathcal{R}_I$. Downsampling introduces aliasing and increases the sampling density by a factor of $|\det M|$, the downsampling factor.

The equivalent input/output Fourier transform relationship is

$$X_d(\omega) = \frac{1}{|\det M|} \sum_{i=0}^{|\det M|-1} X\left((M^T)^{-1}(\omega - 2\pi\mathbf{k}_i)\right) \tag{3.5}$$

where ω is the vector of discrete-time frequency variables and $\mathbf{k}_i$ is a distinct coset vector of M. Note that $\mathbf{k}_0$ is the zero vector (Section 3.3.1 shows how to compute distinct coset vectors). The term $2\pi(M^T)^{-1}\mathbf{k}_i$ (for $i \neq 0$) is called an *aliasing vector*. Aliasing vectors are periodic with a period of 2π in each dimension. A *normalized* aliasing vector is an aliasing vector with the 2π term removed so it is periodic with period 1 in each dimension.

3.1.3 Regular Unimodular Matrices

In one dimension, upsampling or downsampling by a factor of one either has no effect on the data (resampling by $+1$) or reverses the data in time (resampling by -1). Resampling factors of one are, however, very useful in higher dimensions. They correspond to resampling matrices with determinants of ± 1, called regular unimodular resampling matrices [60].

Unimodular Matrix: A *unimodular matrix* is a matrix with determinant of 0, $+1$ or -1.

Regular Unimodular Matrix: A *regular unimodular matrix* is a matrix with a determinant of $+1$ or -1. The inverse of a regular unimodular matrix is also a regular unimodular matrix.

Regular Unimodular Resampling Matrix: A *regular unimodular resampling matrix* is a regular unimodular matrix with integer components.

From the input/output relationships (3.2) and (3.4) for upsamplers and downsamplers, resampling by a regular unimodular matrix does not change the sampling density but does rearrange the input samples. This permutation of input samples is an essential step in decomposing upsampling and downsampling operations into separable form. Since samples are neither deleted nor added when resampling by a regular unimodular matrix, upsampling by such a matrix is equivalent to downsampling by its inverse and vice-versa.

3.1.4 Decomposing A Resampling Matrix Into Smith Form

As mentioned in the introduction, it is often desirable to express a multidimensional multirate operation in separable or decoupled form. One approach is to decompose the resampling matrix underlying the operation. The Smith form of an $m \times n$ integer matrix S is the matrix product $U\Lambda V$, where U and V are square regular unimodular

resampling matrices ($m \times m$ and $n \times n$, respectively) and Λ is an $m \times n$ diagonal matrix whose diagonal elements are non-zero integers [6, 59, 60, 61]. Figure 3.1 shows the U, Λ, and V matrices at each step in the computation of the Smith form. In the first iteration, the element smallest in absolute value is pivoted to the $(1,1)$ entry and all elements in the first column except the diagonal entry are reduced modulo the pivot. When all the entries in the first column other than the diagonal are zero, the process continues with the second column. The first column is left alone, and the next pivot is found and moved to the $(2,2)$ entry. The process continues down the diagonal until the integer matrix has been diagonalized.

For Smith forms of resampling matrices, all component matrices are square and have the same dimension. By decomposing a downsampling matrix M into its Smith form $U_M \Lambda_M V_M$, the downsampling operation becomes equivalent to a shuffling of the input data by U_M followed by a separable downsampling by Λ_M and a reshuffling of the result by V_M. A similar interpretation holds for an upsampling operation.

The following properties hold for the Smith form component matrices U, Λ, and V of a resampling matrix S:

1. If S is diagonal, then a valid choice for U and V is the identity matrix.

2. If S is diagonal, then another choice for U is a permutation matrix. V must permute the columns in the same way that U permutes the rows, so $V = U^{-1}$.

3. When $U = V^{-1}$, which occurs rarely for non-diagonal resampling matrices, the Smith form corresponds to an eigendecomposition yielding integer eigenvalues (the diagonal elements of Λ) with eigenvector matrix U.

4. If S is symmetric, then $U = V^T$.

5. $|\det S| = |\det \Lambda|$.

Because of the last property, the absolute values of the diagonal elements of Λ are integer factors of the absolute value of the determinant of S. If $|\det S|$ is a prime

```
{u, d, v} =
     SmithNormalForm[ resmat,
                      Dialogue -> True ]

         1   0      33   24      1   0

         0   1      27   21      0   1

            pivot location = {2, 2}

         1   0      21   27      1   0

         0   1      24   33      0   1

         modulo reduction by pivot (21)

         1   1      21   6       1   1

         1   0      3    3       1   0

           pivot location = {2, 1}

         1   1      3    3       1   1

         1   0      21   6       1   0

         modulo reduction by pivot (3)

         8   1      3    0       2   1

         7   1      0   -15      1   0


{{{8, 1}, {7, 1}}, {{3, 0}, {0, -15}},
  {{2, 1}, {1, 0}}}
```

Figure 3.1: Iterative Algorithm in the MDSPPs to Compute the Smith Form
The original resampling matrix is the middle matrix in the first two lines of dialogue, i.e. `resmat = {{33, 24}, {27, 21}}`.

$$S = \begin{bmatrix} 736 & 3060 & 1016 \\ 256 & 864 & 308 \\ 424 & 1068 & 428 \end{bmatrix}$$

Smith Form	$\begin{bmatrix} -982 & -7 & -5 \\ -421 & -3 & -2 \\ -841 & -6 & -4 \end{bmatrix}$	$\begin{bmatrix} -4 & 0 & 0 \\ 0 & -180 & 0 \\ 0 & 0 & 24 \end{bmatrix}$	$\begin{bmatrix} 22 & 165 & 47 \\ -70 & -527 & -150 \\ -21 & -158 & -45 \end{bmatrix}$
Smith Canonical Form	$\begin{bmatrix} 982 & 175 & 12 \\ 421 & 73 & 5 \\ 841 & 146 & 10 \end{bmatrix}$	$\begin{bmatrix} 4 & 0 & 0 \\ 0 & 12 & 0 \\ 0 & 0 & 360 \end{bmatrix}$	$\begin{bmatrix} 22 & 165 & 47 \\ -1092 & -8221 & -2340 \\ 511 & 3847 & 1095 \end{bmatrix}$
Alternate Smith Form	$\begin{bmatrix} 14 & 5 & 3 \\ -1 & 11 & 11 \\ -7 & -3 & -2 \end{bmatrix}$	$\begin{bmatrix} 20 & 0 & 0 \\ 0 & 24 & 0 \\ 0 & 0 & 36 \end{bmatrix}$	$\begin{bmatrix} -884 & -2901 & -1045 \\ 5271 & 17311 & 6234 \\ -3558 & -11685 & -4208 \end{bmatrix}$

Figure 3.2: Examples of Different Smith Forms

number, then one diagonal element of Λ must be equal to $\pm \det S$ and the other diagonal elements must be equal to ± 1.

The Smith form of a matrix is not unique. Figure 3.2 shows three different Smith forms for the same resampling matrix. Smith form and Smith canonical form algorithms decompose $m \times n$ integer matrices in $\mathcal{O}(s^4)$ and $\mathcal{O}(s^5)$ elementary arithmetic operations, respectively, where $s = m + n + \log \|S\|_\infty$ such that $\|S\|_\infty = \max_{i,j} |S_{i,j}|$ [60, 62]. (This algorithm analysis assumes that the intermediate integral calculations fit into the machine's integer format.) The Smith canonical form, in which each diagonal element of the Λ matrix is a factor of the next diagonal element, is crucial for an efficient implementation of the multidimensional DFT [6]. For the purposes of simplifying and rewriting multidimensional multirate structures, however, the rules are independent of the Smith form algorithm used.

3.1.5 Finding Common Resampling Matrix Factors

Because the product of resampling matrices does not always commute, resampling matrices have left and right matrix factors as well as left and right matrix multiples [60, 63]. If three resampling matrices A, C, and D exist such that $A = CD$, then

- D is called a *right divisor (factor)* of A,
- C is called a *left divisor (factor)* of A,
- A is called a *left multiple* of D, and
- A is called a *right multiple* of C.

Note that divisor is the term used in the mathematical literature, but for this thesis, factor is usually more descriptive.

For multidimensional multirate structures, a usual operation is to find a *greatest* common matrix factor and a *least* common matrix multiple of two resampling matrices. The common right multiple of two matrices A and B is C when

1. C is a right multiple of A, i.e. $C = AR$ for some resampling matrix R, and
2. C is a right multiple of B, i.e. $C = BS$ for some resampling matrix S.

> **Definition**: The *least* common right multiple (LCRM) of A and B is a right multiple whose determinant is less than or equal to the determinants of all other right multiples of A and B in absolute value.

The LCRM of two matrices is unique only to within a multiplication on the right by a regular unimodular resampling matrix. Associated with the LCRM is the greatest common left divisor (GCLD). A similar definition holds for the least common left multiple (LCLM) and its associated greatest common right divisor (GCRD).

Computing Matrix Factors and Matrix Multiples

The LCRM/GCLD and the LCLM/GCRD of two matrices can be determined numerically. A method for constructing the LCRM of two $n \times n$ non-singular matrices A and B [62, 63] is outlined here. First, form the $2n \times 2n$ matrix

$$\begin{bmatrix} A & B \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$$

where $\mathbf{0}$ denotes an $n \times n$ matrix of all zero entries. Then, by using "integral" Gaussian elimination operations, build a regular unimodular square integer matrix (regular unimodular is defined in Section 3.1.3) X of order $2n$ such that

$$\begin{bmatrix} A & B \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} = \begin{bmatrix} H & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$$

X maps the matrix form formed by concatenating A and B into Hermite normal (lower triangular) form. From the above matrix equation,

$$\mathbf{0} = AX_{12} + BX_{22} \tag{3.6}$$

$$C = AX_{12} = -BX_{22} \tag{3.7}$$

Here, C is the LCRM(A, B) and H is the GCLD(A, B). A dual algorithm can simultaneously find the LCLM and GCRD. For two $n \times n$ matrices A and B, these algorithms require $\mathcal{O}(s^3)$ elementary arithmetic operations where $s = n + \log \|A\|_\infty + \log \|B\|_\infty$ such that $\|S\|_\infty = \max_{i,j} |S_{i,j}|$ [62]. The LCRM, GCLD, LCLM, and GCRD routines in the MDSPPs bear the same name, e.g. `LCRM` and so forth.

Relationship Between Matrix Factors and Lattices

Associated with the LCRM is the greatest common left divisor. For example, the least common multiple of 2 and 3 is 6 but the greatest common divisor of 2 and 3 is one. Similarly, the LCRM of A and B could be ABV (where V is a regular unimodular matrix) so the greatest common left divisor is the identity matrix. This is the

multidimensional analog of relative primeness, which is useful for generalizing commutativity conditions for an up/downsampler in cascade because it relates to finding the greatest common sublattice of two lattices associated with the up/downsampling matrices:

Theorem 2: If $C = \text{LCRM}(A, B)$, then sublattice(C) equals the greatest common sublattice of sublattice(A) and sublattice(B).

Proof: Since C is the LCRM(A, B), $C = AR = BS$ for some resampling matrices R and S. Now applying Theorem 1, we see that sublattice(C) is contained in sublattice(A) and sublattice(B). For every matrix G that is a common right multiple of A and B and C, sublattice(G) is contained in sublattice(A), sublattice(B), and sublattice(C). Now, since the greatest common sublattice of sublattice(A) and sublattice(B) contains all the common sublattices associated with A and B, the proof is complete.

3.2 Resampling Operations in Cascade

In the design of multidimensional multirate filter banks, a fundamental understanding of the tandem connections of upsamplers and downsamplers is often desirable. These properties are already well known for one-dimensional systems [50, 64, 65, 66]. In this section, we simply present the fundamental interconnections of interest. The remaining sections of this chapter develop conditions for simplifying and rearranging these operations.

The simplest interconnection to analyze is upsampling by S followed by downsampling by S. Based on the time domain equations, this tandem processing has no effect on the input signal since $S^{-1}S = I$,

As mentioned in the introduction, resampling matrices do not generally commute. Upsampling by L_1 followed by upsampling by L_2 is equivalent to upsampling by

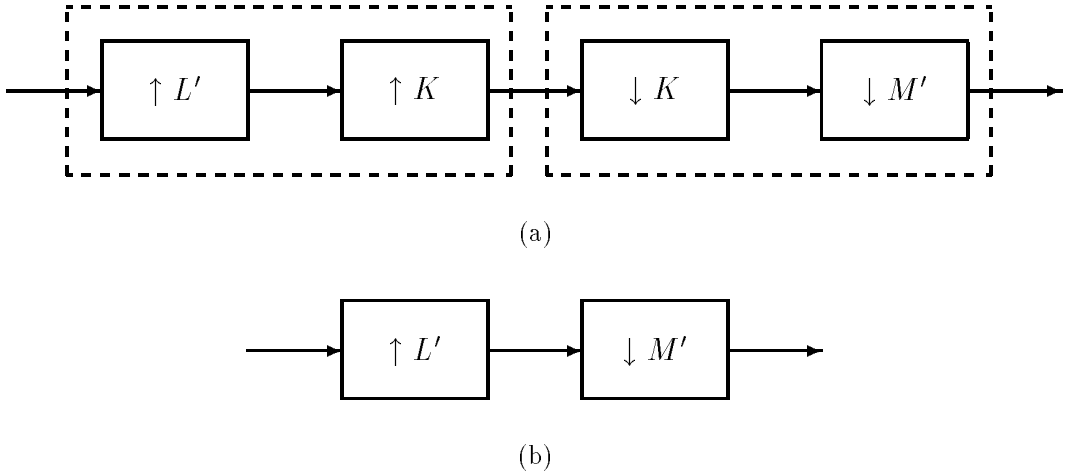


Figure 3.3: Equivalent Structures for $\uparrow L = KL'$ Followed by $\downarrow M = KM'$

L_2L_1 because according to equation (3.2), it is the inverse of the upsampling matrix

$$(L_2L_1)^{-1} = L_1^{-1}L_2^{-1}$$

that determines how the samples from the input signal are mapped to the output signal (so the order of operations is switched). It is also easy to prove that downsampling by M_1 followed by downsampling by M_2 is equivalent to downsampling by M_1M_2. Note that in general, the cascade of two upsamplers (or two downsamplers) is not commutative. They are commutative if and only if the matrix products commute, i.e. $L_1L_2 = L_2L_1$ for upsamplers or $M_1M_2 = M_2M_1$ for downsamplers.

Because resampling matrices do not commute, resampling matrices have left and right matrix factors. In one dimension, the common factors in an up/downsampling cascade can be removed. In higher dimensions, this condition can be defined in terms of common left matrix factors. Consider the system shown in Figure 3.3 which downsamples by $M = K\,M'$ after upsampling by $L = K\,L'$. Since $\uparrow KL'$ is equivalent to $\uparrow L'$ followed by $\uparrow K$, we see that the system in Figure 3.3a can be simplified to the system in Figure 3.3b. In other words, if we can factor the two resampling matrices

such that they share a common matrix factor on the left, then that common factor can be removed without changing the overall system response.

3.3 The Role of Smith Form Matrices

The Smith form decomposition of a matrix has been useful in many aspects of multidimensional signal processing including the design of multidimensional multirate filter banks [59, 61, 67] and the design of multidimensional DFT and FFT algorithms [6]. Furthermore, the Smith form decomposition has been used in other areas of electrical engineering including control theory for linear systems [39]. In this section, we present new applications of the Smith form decomposition. The first is the computation of distinct coset vectors (Section 3.3.1) which can generate the aliasing vectors associated with a downsampling matrix (see Section 3.1.2). Aliasing vectors play a role in determining the commutativity of up/downsampler cascades in Section 3.4. The second new application of Smith forms in this section is the finding of the greatest common sublattice (GCS) of two lattices (Section 3.3.2). The GCS leads to rules for simplifying cascades of resampling operations (Section 3.3.3). Later in Section 3.4, the GCS will be key in finding a second set of commutativity conditions for up/downsamplers in cascade.

3.3.1 Computation of Coset Vectors

The distinct coset vectors of resampling matrix S are those vectors drawn from the origin to the points inside the fundamental parallelepiped of S (the points always have non-negative values). In one dimension, the parallelepiped is the block of samples being reindexed. In the separable multidimensional case, S is diagonal and the fundamental parallelepiped is a rectangular prism whose "lower left-hand" corner is the origin and whose "upper right-hand" corner is $(p_1 - 1,\, p_2 - 1,\, \ldots)$ where $p_i = |S_{i,i}|$. Clearly, the ease in enumerating points inside a rectangular prism simplifies the com-

putation of the distinct coset vectors for diagonal resampling matrices.

For more general resampling matrices, it is not as easy to compute the distinct coset vectors because it is harder to enumerate the points in the fundamental parallelepiped. The following theorem, offered by Gardos [68], handles the case where the resampling matrix S is the product of a regular unimodular matrix U and a diagonal matrix Λ. Although the theorem states that all of the diagonal elements of Λ must be positive, this actually covers the case for all integer diagonal matrices Λ because V can always be adjusted so that Λ has only positive diagonal elements. The proof is given in the appendix of [72].

> **Theorem 3:** Let U be a unimodular integer matrix and Λ be a diagonal matrix with all positive elements, then $\mathrm{FPD}(U\Lambda) = U\ \mathrm{FPD}(\Lambda)$.

For a general resampling matrix S, the points inside of the fundamental parallelepiped can be enumerated by first decomposing S into Smith form $U\Lambda V$. Then, if any of the diagonal elements of Λ are negative, multiply Λ on the right and V on the left by the diagonal matrix D such that $D_{i,i} = \mathrm{sgn}(\Lambda_{i,i})$. As mentioned in the beginning of this section, it is easy to calculate the distinct coset vectors for Λ because it is diagonal. From Theorem 3, $\mathrm{FPD}(U\Lambda)$ is just the mapping of $\mathrm{FPD}(\Lambda)$ by U. From Corollary 1.1, since V is unimodular, $\mathrm{sublattice}(S) = \mathrm{sublattice}(U\Lambda)$, so

$$\mathrm{FPD}(S) \;=\; \mathrm{FPD}(U\Lambda) \;\bmod\; S \;=\; U\,\mathrm{FPD}(\Lambda) \;\bmod\; S$$

Therefore, the points in the fundamental parallelepiped of S become the points in the fundamental parallelepiped of Λ mapped by U modulo S:

$$\mathrm{FPD}(S) = \{((U\,\mathbf{l}))_S \mid \mathbf{l} \in \mathrm{FPD}(\Lambda)\} \tag{3.8}$$

where $((\cdot))_M$ is vector-matrix modulo operator defined as

$$((\mathbf{x}))_M \;=\; \mathbf{x} \;-\; M\,\lfloor M^{-1}\mathbf{x} \rfloor$$

such that the $\lfloor\cdot\rfloor$ operator takes the floor of every component of the vector argument.

For a given $n \times n$ resampling matrix S, computing (3.8) requires calculating S^{-1} and U. With $s = n + \log \|S\|_\infty$, computing (3.8) requires one Smith form decomposition ($\mathcal{O}(s^4)$) plus one matrix inversion ($\mathcal{O}(n^3)$) plus three matrix-vector operations, one integer vector subtraction, and one vector floor operation per point of the fundamental parallelepiped of Λ. Since the fundamental parallelepiped contains $|\det S|$ points, equation (3.8) requires $\mathcal{O}(s^4 + n^3 + n^2|\det S|)$ multiplications plus $\mathcal{O}(n^2|\det S|)$ floor operations. When integral resampling occurs in every dimension, i.e. when none of the diagonal elements are ± 1, the n^3 term can be ignored because $|\det S| \geq 2^n \gg n$, so the matrix inversion operation is "free". The n^4 term can only be ignored when $|\det S| \gg n^2$ which is satisfied when the number of dimensions rises above 7 and integral resampling occurs in every dimension. For image and video processing, the number of dimensions n is 2 or 3 so the n^4 term is significant. Efficient algorithms to compute cosets that do not use the Smith form are discussed in [69].

3.3.2 Computing the Greatest Common Sublattice

The least common right multiple provides one way to compute the greatest common sublattice (GCS) associated with two resampling matrices (see Section 3.1.5). Sometimes, the Smith form decomposition can be used to find the GCS.

The GCS associated with two diagonal resampling matrices is trivial to compute. Namely, given two diagonal resampling matrices Λ_1 and Λ_2, compute a new diagonal matrix Λ whose ith diagonal element is the greatest common divisor of the ith diagonal element of Λ_1 and Λ_2. It is easily verified that the greatest common sublattice associated with Λ_1 and Λ_2 is simply the sublattice associated with the resampling matrix $\Lambda_1'\Lambda_2'$ where $\Lambda_1 = \Lambda\Lambda_1'$ and $\Lambda_2 = \Lambda\Lambda_2'$.

For non-diagonal resampling matrices S_1 and S_2, the GCS can be found when the two resampling matrices share a common left regular unimodular matrix factor U. If the decompositions share a common left factor $U = U_1 = U_2$, then according to Corollary 1.1 finding the GCS associated with S_1 and S_2 is equivalent to finding

the GCS associated with $S_1' = U\Lambda_1$ and $S_2' = U\Lambda_2$. This is easily done by noting that U is a one-to-one and onto linear operator. Hence, one can simply compute the GCS associated with the resampling matrices Λ_1 and Λ_2 and then apply the inverse mapping U^{-1} to generate the GCS associated with S_1 and S_2.

In order to compute the GCS using this approach, we introduce the following algorithm to decompose two resampling matrices S_1 and S_2 into similar Smith forms:

1. Find a Smith form of $S_1 = U_1\Lambda_1 V_1$.
2. Set $U_2 = U_1$
3. Find Λ_2 and V_2 from the relationship $U_1^{-1}S_2 = \Lambda_2 V_2$
   (a) $[\Lambda_2]_{i,i}$ is the gcd of the elements of row i of the matrix $U_1^{-1}S_2$.
   (b) V_2 can be computed using $V_2 = \Lambda_2^{-1}\left[U_1^{-1}S_2\right]$.
4. S_1 and S_2 share a common U factor if V_2 is a regular unimodular resampling matrix

Any Smith form algorithm can be used in the first step. For $n \times n$ resampling matrices S_1 and S_2, the algorithm requires $\mathcal{O}\left([n + \log \|S\|_\infty]^4 + n^3\right)$ elementary arithmetic operations and $\mathcal{O}(n^2)$ gcd operations. The $\mathcal{O}(n^3)$ term arises from the computation of $U_1^{-1}S_2$. Using a dual method, a Smith form decomposition can be generated for $V_1 = V_2$. In the MDSPPs, the routines `SmithFormSameU` and `SmithFormSameV` implement these two algorithms.

3.3.3 Simplification of Resamplers in Cascade

Armed with the ability to compute Smith forms and the GCS, we address the issue of simplying and rearranging upsamplers and downsamplers in cascade. Consider the up/downsampler cascade in Figure 3.4a, where we have expressed $M = U_M\Lambda_M V_M$ and $L = U_L\Lambda_L V_L$ in their Smith forms. If the decomposition can be performed using

the same left unimodular matrix $U_M = U_L = U$ (see Section 3.3.2), then it is trivial to remove a factor of U from each operator and the resulting system is shown in Figure 3.4b. Now, downsampling by Λ_M follows upsampling by Λ_L so common factors between the ith diagonal elements of Λ_M and Λ_L can be removed thereby creating two new diagonal matrices Λ'_M and Λ'_L shown in Figure 3.4c. Since Λ'_L and Λ'_M are coprime, upsampling by Λ'_L and downsampling by Λ'_M can be switched (Figure 3.4d). Figure 3.4e combines the remaining four operations into two by using the fact that downsampling by a regular unimodular matrix is equivalent to upsampling by its inverse and vice-versa (Section 3.1.3). Hence, Figure 3.4 shows that sometimes the operations in an up/downsampler cascade can be switched by modifying the resampling matrices. In a similar way, Figure 3.5 shows that the cascade of a downsampler followed by an upsampler can sometimes be simplified and even rearranged.

3.4 Commutativity of Resamplers in Cascade

We now derive two equivalent sets of conditions for the commutativity of downsampling by M and upsampling by L in cascade. In the frequency domain, the conditions are derived by comparing the frequency response of an up/downsampler cascade with that of a down/upsampler cascade (see equations (3.3) and (3.5) in Section 3.1.2). For the two-dimensional case, the conditions are [70]:

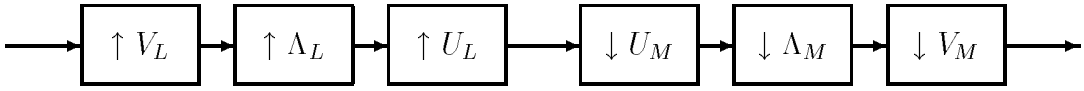
FD1 The products of the resampling matrices commute, $LM = ML$, and

FD2 The two sets of normalized aliasing vectors $\mathbf{N}_{\text{up/down}}$ and $\mathbf{N}_{\text{down/up}}$ are equivalent such that

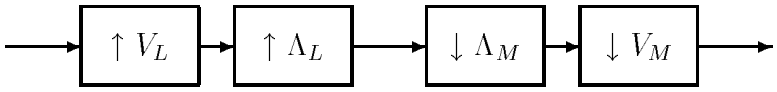
$$\mathbf{N}_{\text{up/down}} = \{\left[M^T\right]^{-1} \mathbf{k}_i,\ i = 0 \ldots |\det M| - 1\}$$

$$\mathbf{N}_{\text{down/up}} = \{L^T \left[M^T\right]^{-1} \mathbf{k}_i,\ i = 0 \ldots |\det M| - 1\}$$

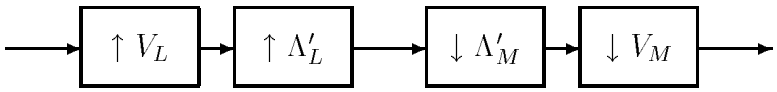
These conditions were derived without reference to the Smith form decomposition. The Smith form decomposition of the downsampling matrix M enables the enumer-



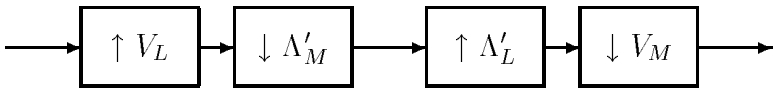
(a) Cascade in Smith form



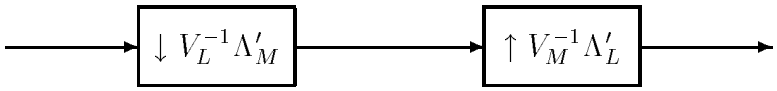
(b) Simplified cascade *if* $U_M = U_L$



(c) Simplified cascade from (b) with common factors removed

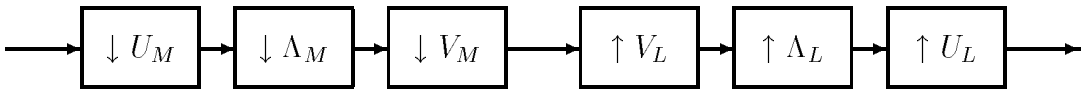


(d) Reversing the order of operations in (c) since Λ'_L and Λ'_M are coprime

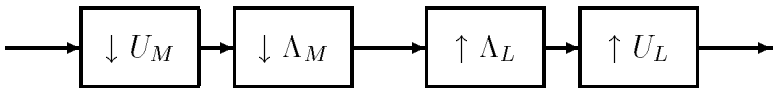


(e) Combining operations in (d) so that downsampling comes first

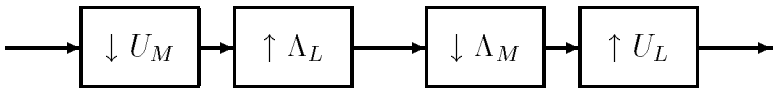
Figure 3.4: Five Equivalent Forms of an Up/downsampler Cascade



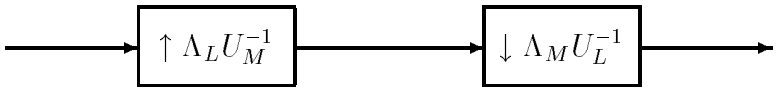
(a) Cascade in Smith Form



(b) Simplified cascade *if* $V_M = V_L$



(c) Reversing order of operations in (b) *if* Λ_M and Λ_L are coprime



(d) Combining operations in (c)

Figure 3.5: Four Equivalent Forms of a Down/upsampler Cascade

ation of the coset vectors k_i (see Section 3.3.1); for an $n \times n$ matrix M, generating the coset vectors requires $\mathcal{O}\left([n + \log \|M\|_\infty]^4 + n^3 + n^2 |\det M|\right)$ multiplications plus $\mathcal{O}(n^2 |\det M|)$ floor operations. The coset vectors can then be used to calculate the two sets of normalized aliasing vectors $\mathbf{N}_{\text{up/down}}$ and $\mathbf{N}_{\text{down/up}}$ each having $|\det M|$ elements. If the two sets are identical, then the cascade commutes. Thus, the Smith form is the key that unlocks the commutativity conditions (based on equivalence in the frequency domain) for the general multidimensional case.

In the time domain, we derive the conditions by comparing the time response of an up/downsampler cascade with that of a down/upsampler cascade (see equations (3.2) and (3.4) in Section 3.1.2). We discovered the time domain conditions [71, 72] in parallel with [52]. The two resampling operations are commutative if and only if

TD1 $ML = LM$, and

TD2 $\text{LCRM}(M, L) = MLV$ where V is a regular unimodular matrix.

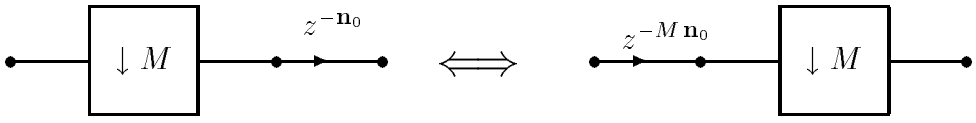
There are at least two ways to test the last condition **TD2**. The first is to see if the matrix product of the inverse of (ML) and $\text{LCRM}(M, L)$ yields a regular unimodular integer matrix. A second method checks to see if the greatest common left divisor (GCLD) of M and L is a regular unimodular matrix, which is faster because it does not require the matrix inversion of the first method. In either method, the LCRM of $n \times n$ matrices M and L requires $\mathcal{O}(s^3)$ elementary arithmetic operations where $s = n + \log \|M\|_\infty + \log \|L\|_\infty$ [62].

A time-domain and a frequency-domain analysis leads to similar pairs of commutativity conditions for an up/downsampler cascade, but the time-domain conditions require far fewer computations. Commutativity of matrix products, the first condition, is easy to check because it only requires $\mathcal{O}(n^3)$ additions and multiplications for $n \times n$ resampling matrices. Of the second commutativity conditions **TD2** and **FD2**, **TD2** (the one based on the time domain) is much simpler to check because the order of computations is independent of the determinant of the downsampling matrix and varies with n^3 instead of n^4.

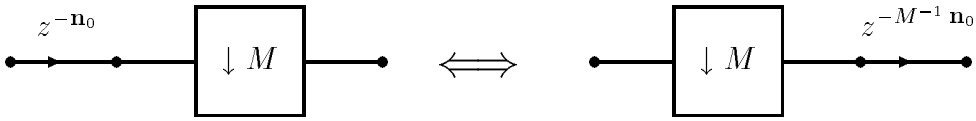
3.5 A Comprehensive Set of Rules

In this section, we collect many rules for multidimensional multirate signal processing. The rules are presented as figures. Figures 3.6 and 3.7 are generalizations of Figures 3.8 and 3.9 in [49], and Figure 3.8 is a generalization of the interaction between LTI filters and up/downsamplers discussed in Section 3.4 of [49]. Figure 3.9 summarizes the rules for cascades of resamplers from Section 3.2 which are adaptations of the identities in Figure 3.10 of [49].

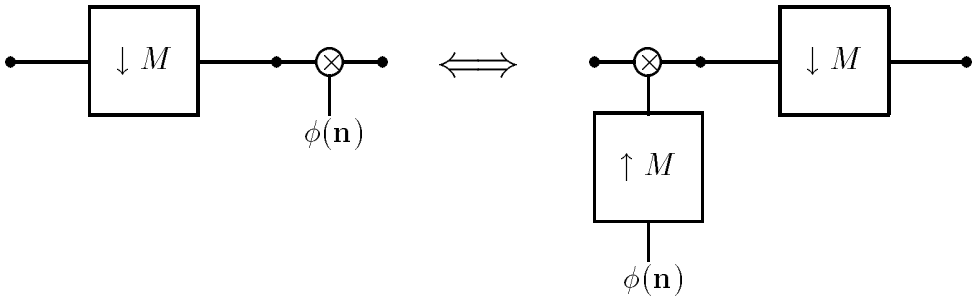
The one-dimensional rules governing the commutativity of resamplers in cascade rely on the fact that integer multiplication commutes and that factoring is complete over the integers. In one dimension, two upsamplers (or two downsamplers) in cascade always commute, but in multiple dimensions, the operations can be switched only if the product of the two resampling matrices commutes, as shown in Figure 3.10a and 3.10b. An up/downsampler in one dimension commutes if the resampling factors are relatively prime, i.e. if the least common multiple is the product of the two factors, so this can be easily extended to separable up/downsampling where the resampling matrices are diagonal (Figure 3.10c). For non-separable resampling, the two conditions for commutativity derived in Section 3.4 are that the product of the resampling matrices commutes and that the least common right multiple of the two resampling matrices equals the product of the two matrices and a regular unimodular matrix (Figure 3.10d). An alternate second condition is that the two sets of aliasing vectors, one for the cascade and one for the commuted cascade, are equivalent (Figure 3.10e). The time-domain conditions, however, require far fewer computations.



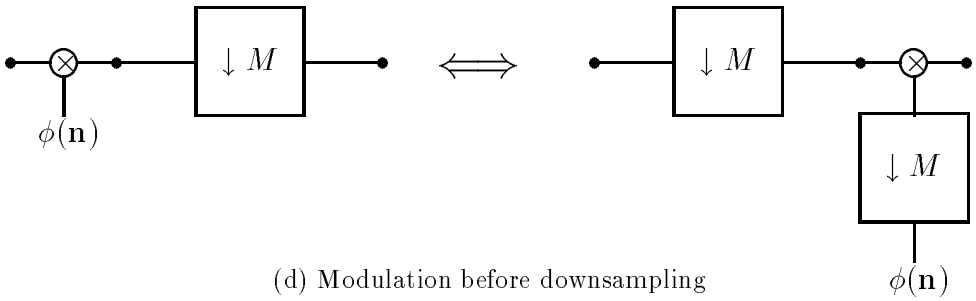
(a) Delay after downsampling



(b) Delay before downsampling

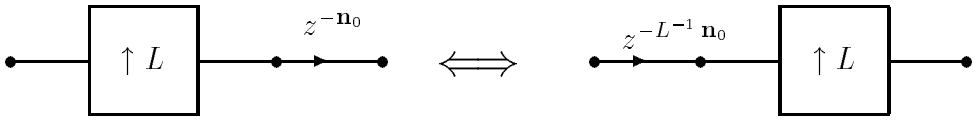


(c) Modulation after downsampling

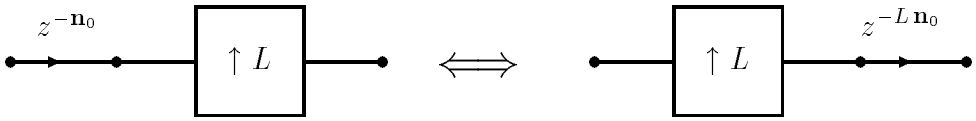


(d) Modulation before downsampling

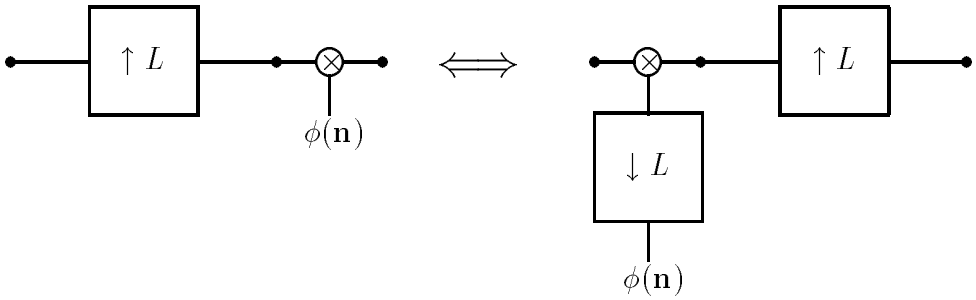
Figure 3.6: Identities for Downsamplers



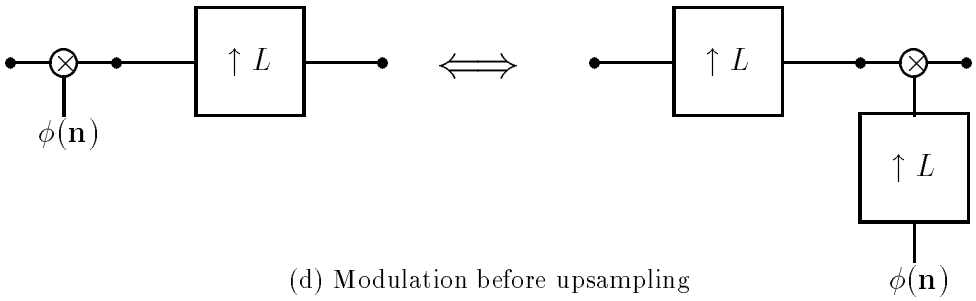
(a) Delay after upsampling



(b) Delay before upsampling

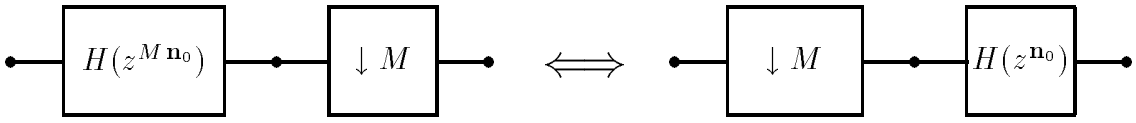


(c) Modulation after upsampling

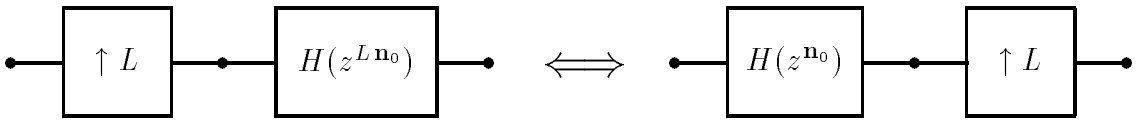


(d) Modulation before upsampling

Figure 3.7: Identities for Upsamplers



(a) Filtering before downsampling [52, 61]

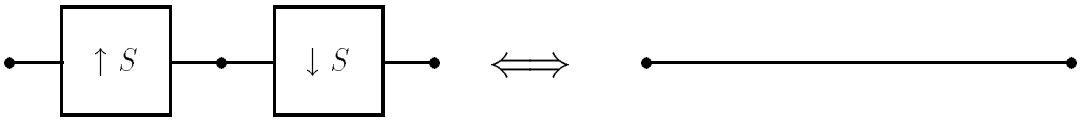


(b) Filtering after upsampling [52, 61]

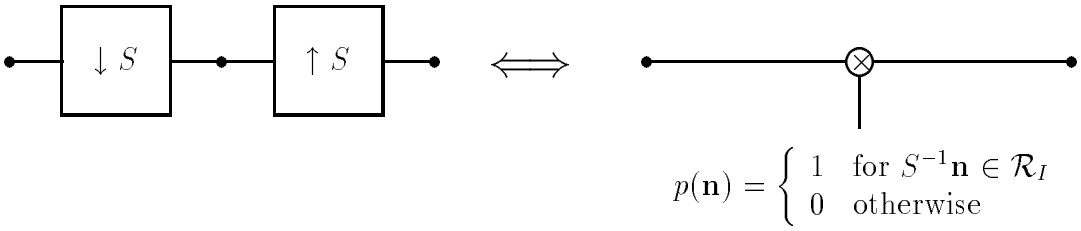
Figure 3.8: Interactions between Up/downsamplers and LTI Filters

In multidimensional signal processing, one typically encounters situations that do not arise in one dimension. This is the case for decomposing matrices into Smith form. The properties of resampling operations expressed in Smith form (with component diagonal and regular unimodular matrices) are summarized in Figure 3.5. Figure 3.12 shows how the Smith form can be used to remove redundancy (if any) from up/downsampler cascades. Figure 3.13 identifies the conditions for which up/downsampler cascades do not commute but yet the operations can be switched by altering the resampling matrices.

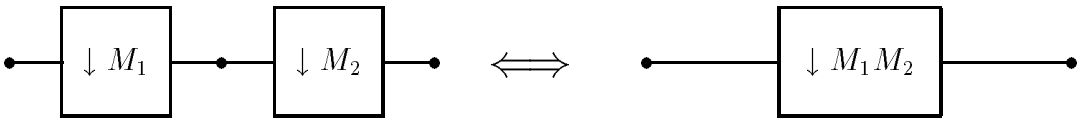
The last figure, Figure 3.14, shows what happens when a shifter occurs between an upsampler and a downsampler. Remembering that one-dimensional downsampling in terms of blocks of samples is analogous to multidimensional downsampling in terms of fundamental parallelepipeds of samples, the shifter will either shift samples off of the downsampling lattice (Figure 3.14a) or it will delay the incoming samples an integer number of parallelepipeds (Figure 3.14b). Figure 3.14c shows a way to decompose a shift operation that exists between an upsampler and downsampler in cascade. In



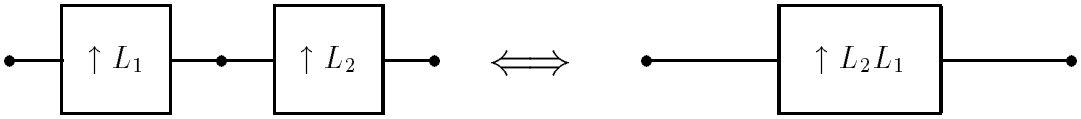
(a) Up/downsampling by the same matrix



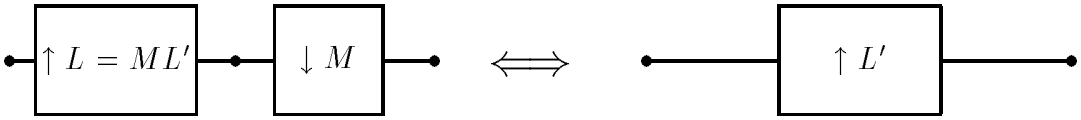
(b) Down/upsampling by the same matrix



(c) Cascade of downsamplers

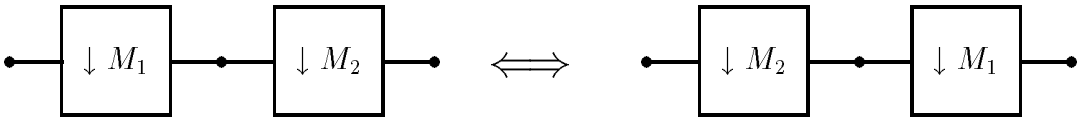


(d) Cascade of upsamplers

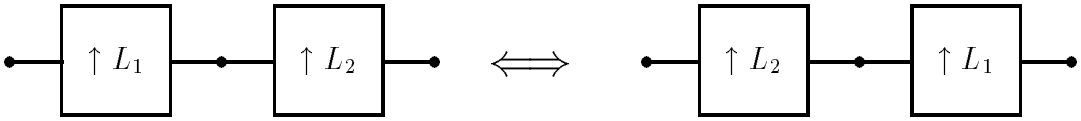


(e) If $L' = M^{-1}L$ is a non-singular integer matrix

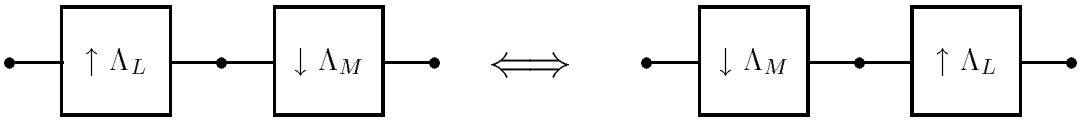
Figure 3.9: Identities for Cascades of Upsamplers and Downsamplers



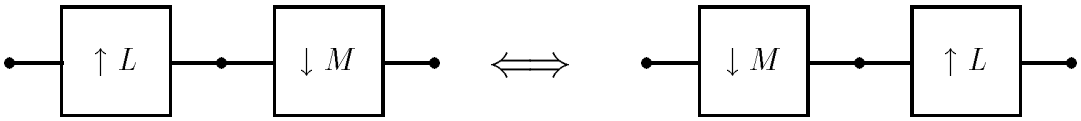
(a) if $M_1 M_2 = M_2 M_1$ (also reported in [52])



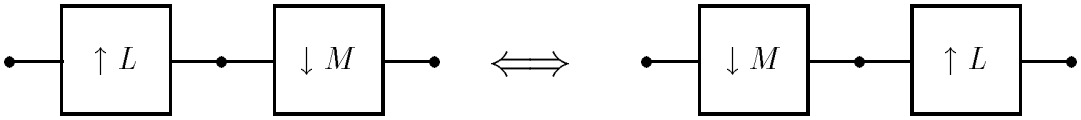
(b) if $L_1 L_2 = L_2 L_1$ (also reported in [52])



(c) if Λ_L and Λ_M are diagonal resampling matrices whose corresponding elements are relatively prime

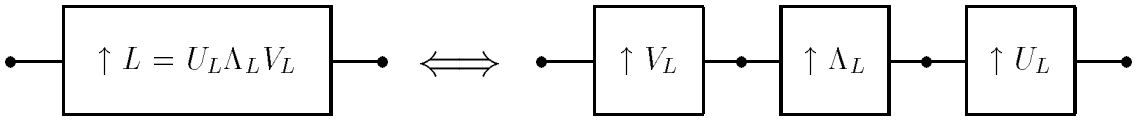


(d) if $LM = ML$ and $\mathrm{LCRM}(L, M) = LMV$ such that V is a regular unimodular resampling matrix (see Section 3.4; independently reported in [52])

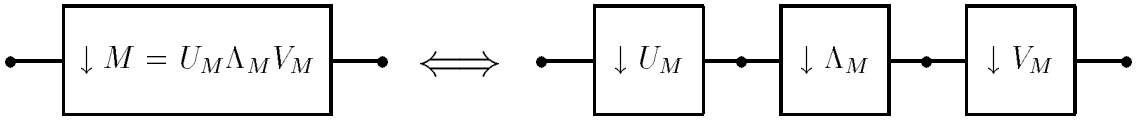


(e) if $LM = ML$ and the two sets of aliasing vectors are equivalent (see Section 3.4; adapted from [70])

Figure 3.10: Commutativity of Cascades of Upsamplers and Downsamplers

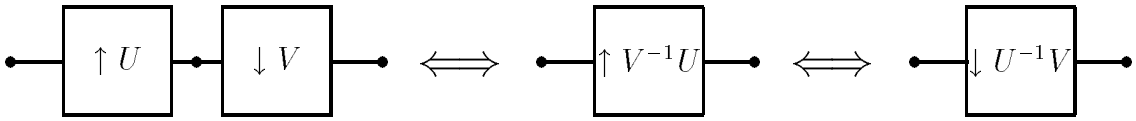


(a) The Smith form of an upsampler

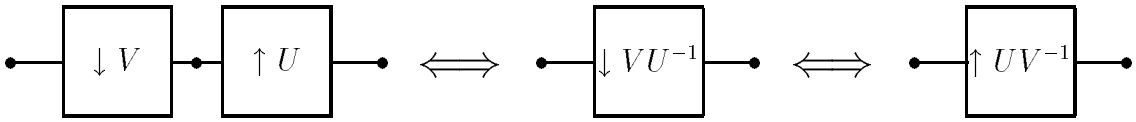


(b) The Smith form of a downsampler

(c) Resampling by regular unimodular matrices

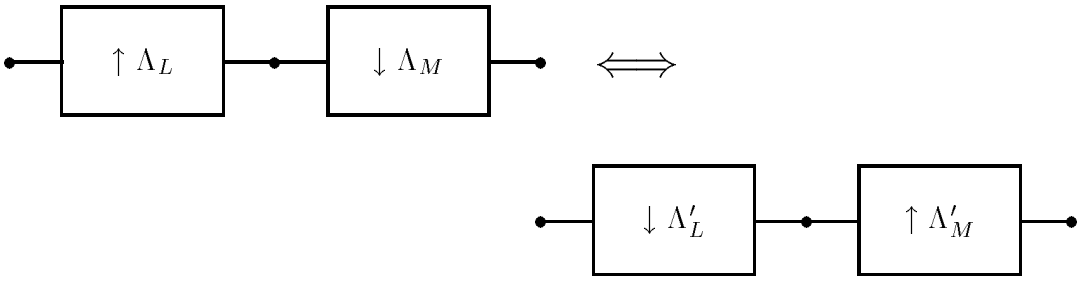


(d) Up/downsampling by regular unimodular matrices

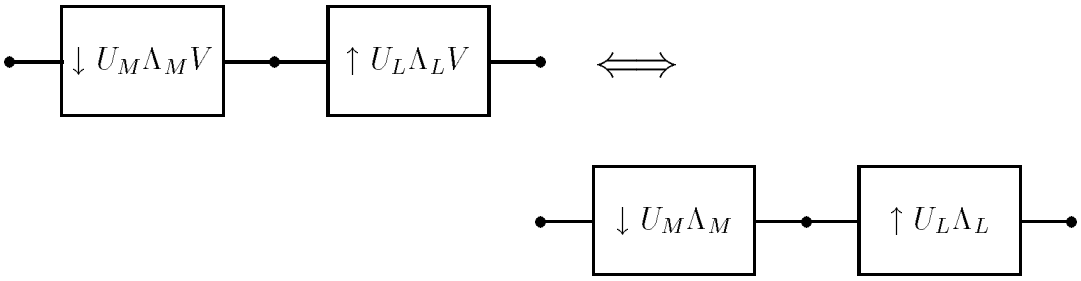


(e) Down/upsampling by regular unimodular matrices

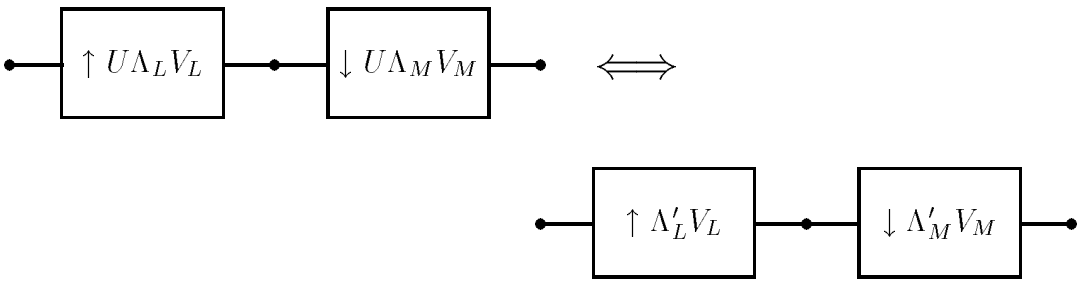
Figure 3.11: Fundamental Identities Based on the Smith Form Decomposition
The Smith form decomposition rewrites a resampling matrix into a matrix product $U\Lambda V$ where U and V are regular unimodular matrices and Λ is a diagonal matrix (see Section 3.3).



(a) If Λ_L and Λ_M are diagonal resampling matrices, then common factors between each corresponding pair of diagonal elements can be removed.



(b) If the resampling matrices can be decomposed into Smith form using the same right unimodular matrix, then that resampling operation cancels (see Section 3.3.3).

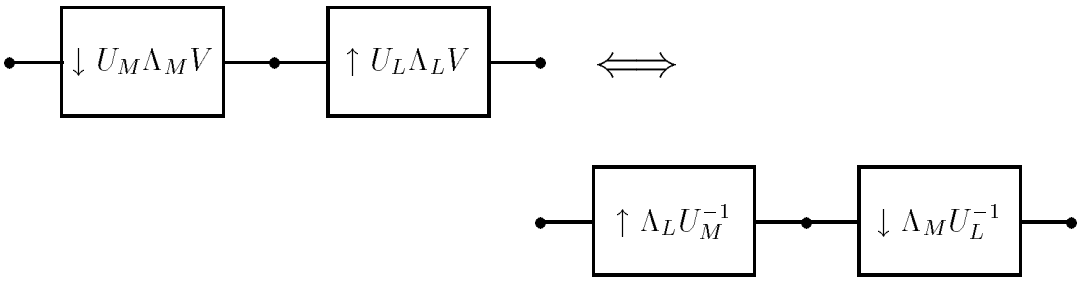


(c) If the resampling matrices can be decomposed into Smith form using the same left unimodular matrix, then that resampling operation cancels as well as any common factors between each corresponding pair of diagonal elements (see Section 3.3.3).

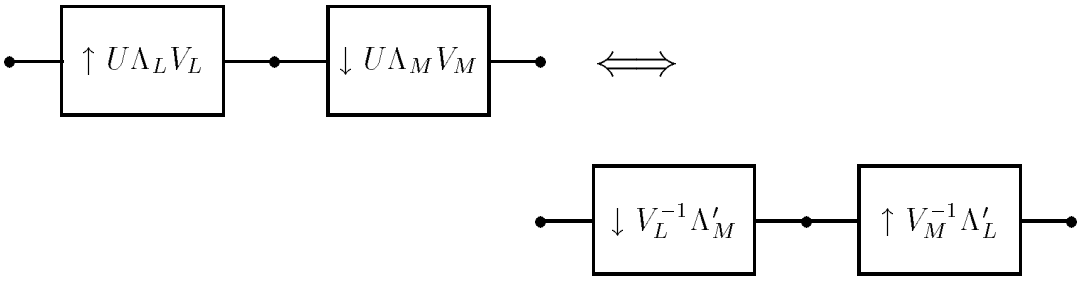
Figure 3.12: Removing Redundancy in Cascades of Upsamplers and Downsamplers

one dimension, given two non-zero integers a and b, a family of solutions for integers α and β in $\alpha\, a \,+\, \beta\, b \;=\; \gcd(a, b)$ can always be found. An efficient algorithm to find α and β first converts the equation to a Bezout identity by dividing out $\gcd(a, b)$ to produce $\alpha\hat{a} + \beta\hat{b} = 1$ and then enumerates either α from 0 to $|\hat{b}| - 1$ if $|\hat{b}| < |\hat{a}|$ or β from 0 to $|\hat{a}| - 1$ otherwise until an integral solution is found for the other variable. (The Bezout identity, solved by the `BezoutNumbers` routine in the MDSPPs, plays a key role in transforming Smith forms into canonical form [60].) In multiple dimensions, given two square non-singular integer matrices M and L, a family of integer vectors $\mathbf{n}_M$ and $\mathbf{n}_L$ can always be found to satisfy $M\,\mathbf{n}_M + L\,\mathbf{n}_L = \mathbf{n}_0$. First, multiply both sides of this equation on the left by the inverse $\mathrm{GCRD}(M, L)$ to convert it into a multidimensional Bezout identity, $\hat{M}\,\mathbf{n}_M \,+\, \hat{L}\,\mathbf{n}_L \;=\; \mathrm{GCRD}^{-1}(M, L)\;\mathbf{n}_0$. Next, enumerate either the coset vectors of $\hat{L}$ as the candidates for $\mathbf{n}_M$ if $|\det L| < |\det M|$ or the coset vectors of M as the candidates for $\mathbf{n}_L$ otherwise until an integer vector solution is found for the other variable. Like the one-dimensional Bezout identity, the answers for $\mathbf{n}_M$ and $\mathbf{n}_L$ are periodic. The family of solutions can be obtained by shifting $\mathbf{n}_M$ by L times some integer vector $\mathbf{l}$ and by shifting $\mathbf{n}_L$ by $-M\mathbf{l}$. In the MDSPPs, the `EuclidFactors` routine finds $\mathbf{n}_M$ and $\mathbf{n}_L$ in the relation $M\,\mathbf{n}_M \,+\, L\,\mathbf{n}_L \;=\; \mathbf{n}_0$ for relatively prime M and L.

For completeness, we include the polyphase decomposition of a multidimensional rational rate changer [51, 52]. Like the one-dimensional version [49], the polyphase decomposition resamples the filter coefficients to create a more efficient implementation of the rate changer. Just as in the one-dimensional case, the polyphase form exists whenever the upsampling and downsampling matrices are relatively prime [52]. The polyphase decomposition is diagramed in Figure 3.15.

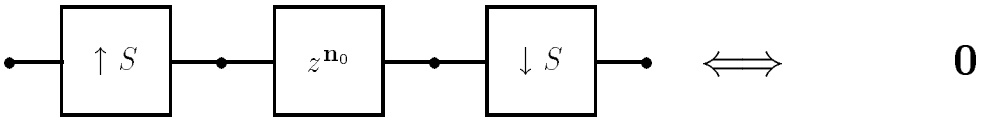


(a) Switching operations in a non-commutable down/upsampler cascade if the diagonal matrices Λ_M and Λ_L are relatively prime

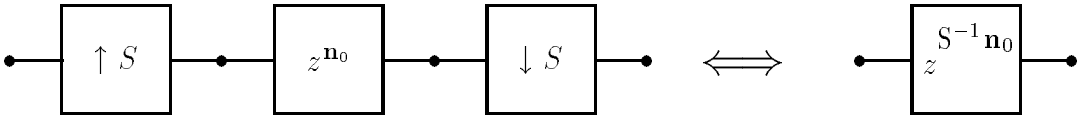


(b) Switching operations in a non-commutable down/upsampler cascade by removing the common factors of the corresponding diagonal elements of Λ_L and Λ_M to produce Λ'_L and Λ'_M.

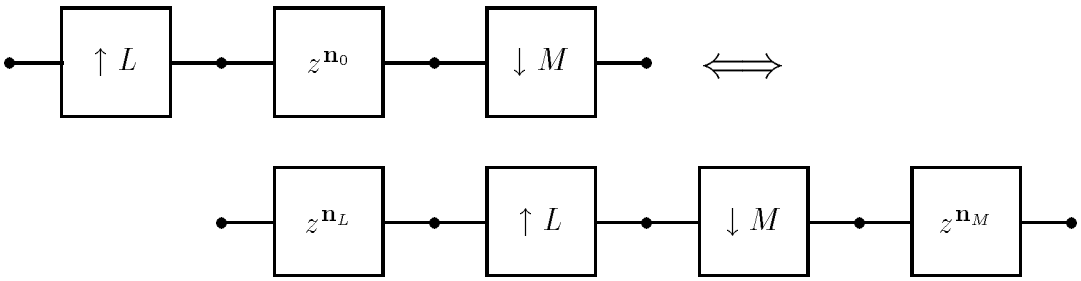
Figure 3.13: Switching Operations in Non-commutable Cascades



(a) Up/downsampling by S when the shift vector $\mathbf{n}_0 \notin$ sublattice(S) or equivalently when $S^{-1}\mathbf{n}_0$ is not an integer vector

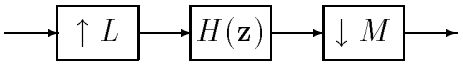


(b) Up/downsampling by S when the shift vector $\mathbf{n}_0 \in$ sublattice(S) or equivalently when $S^{-1}\mathbf{n}_0$ is an integer vector

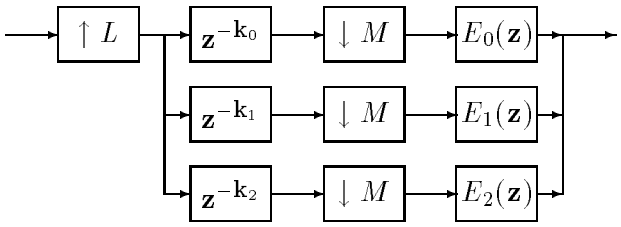


(c) For resampling matrices L and M, $\mathbf{n}_0$ can always be rewritten as $\mathbf{n}_0 = L\mathbf{n}_L + M\mathbf{n}_M$ using a form of Euclid's algorithm or equivalently by converting the equation to a Bezout identity

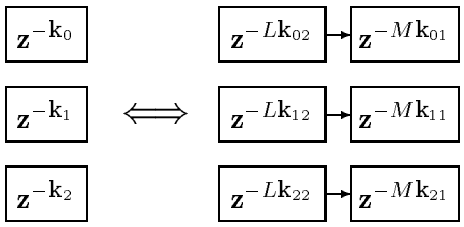
Figure 3.14: Interaction between Up/downsampler Cascades and Shifters



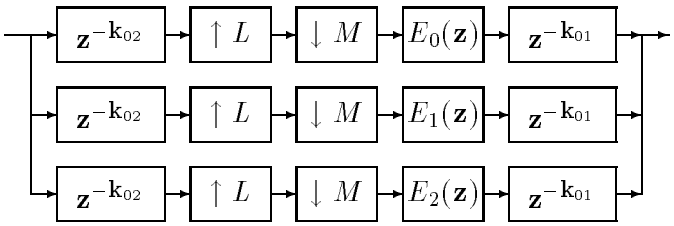
(a) Multidimensional Rational Decimation System



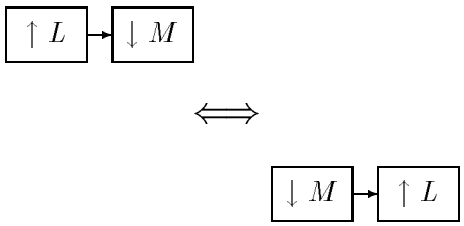
(b) Rewriting the Downsampling Stage



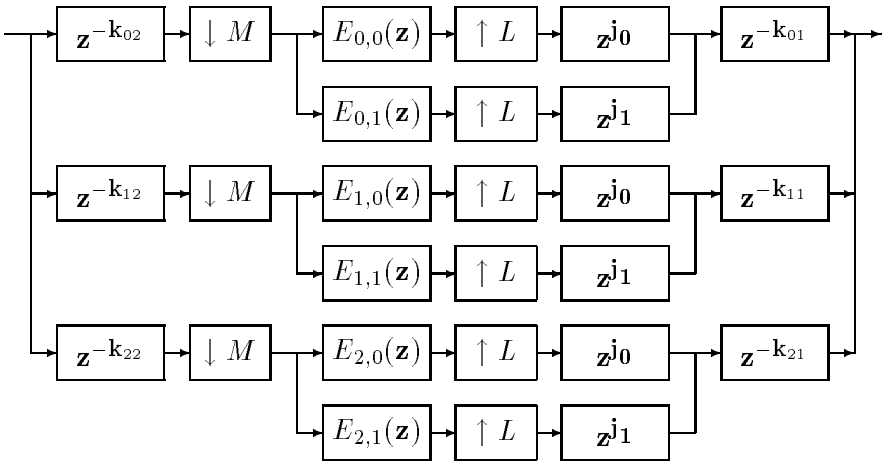
(c) Rewriting the Shift Operations (if L and M are relatively prime on the left)



(d) Moving the Shift Operations Outside of the Up/Downsamplers



(e) Switching the Order of the Up/Downsamplers (if $LM = ML$ and L and M are relatively prime)



(f) Final Polyphase Form

Figure 3.15: Polyphase Implementations of Rational Decimation Systems In the end, all filtering operations are performed at the lowest sampling rate in the system (adapted from [51]).

3.6 Summary

As shown in Figures 3.6 through 3.14, this chapter presents original research that generalizes the simplification and rearrangement rules for one-dimensional multirate signal processing in [49] to multiple dimensions. This chapter also generalizes two other important one-dimensional rules to multiple dimensions— (1) the commutativity of an upsampler and downsampler in cascade, and (2) pulling out the shift operation in the cascade of an upsampler, shifter, and downsampler. Vaidyanathan and Chen [52, 61] independently derived these two rules in multiple dimensions and then applied them to find polyphase implementations of multidimensional rate changers (i.e. rational decimation systems). We include their polyphase rules for completeness.

This chapter also develops the algorithms necessary to evaluate the conditions of the rules in Figures 3.9, 3.10, 3.12, and 3.13. One of new algorithms computes the greatest common sublattice by computing common left matrix factors using on Smith form decompositions. Its dual, which finds common right matrix factors, is useful in removing redundant operations and switching operations in non-commutable cascades. The new algorithm underlying Figure 3.10e uses the Smith form decomposition of the downsampling matrix to compute the two sets of aliasing vectors.

All of the supporting algorithms and all of the rules in Figures 3.6 through 3.14 have been encoded in the MDSPPs. "LatticeTheory.m" contains the algorithms and "RewriteRules.m" contains the rules. A version of "LatticeTheory.m" is available that does not rely on other routines in the MDSPPs. Also encoded in our lattice theory package is a family of Smith form algorithms that find Λ matrices whose diagonal elements are as close to one another in absolute value as possible and/or minimize the norm of either U or V [27]. Other researchers have used this new family of Smith form algorithms in our algorithm to compute common left matrix factors as a part of a procedure to design multidimensional non-uniform filter banks [73].

CHAPTER 4

The Signal Processing Packages: Implementing Linear Systems Theory in *Mathematica*

The multidimensional signal processing packages (MDSPPs) [1, 2, 3] represent a significant step in implementing linear systems *theory* in a computer algebra program (*Mathematica*). *Theory* refers to the algebraic and other symbolic operations used in studying the behavior of linear systems. The MDSPPs obey *Mathematica*'s syntax rules and follow engineering convention when possible. The MDSPPs define many new signals (Section 4.1) and systems (Section 4.2) that are missing in the core of *Mathematica*. Section 4.3 and 4.4 introduce some of the plotting and symbolic abilities of the MDSPPs. Chapter 5 covers more advanced features of the MDSPPs.

4.1 Defining Signals

The MDSPPs define the signals listed in Table 4.1. The new signals are functions that take one argument, except for `Pulse` and `CPulse` which require two— the pulse length and the variable plus the offset (if any). One-dimensional signals require a variable or expression to specify the domain of the signal. For example, as a function of t, the continuous step function u_{-1} would take the form of `Unit[-1][t]` or more simply `CStep[t]` which stands for "Continuous Step". The expression `CStep[t-2]` would represent a step function delayed by 2 or $u_{-1}(t-2)$. Signals can appear in algebraic expressions, such as `t CStep[t]`, which is one way to write the ramp function.

Object	Meaning
`CPulse`	continuous pulse: $$\sqcap_L(t) = \begin{cases} 0 & t < 0 \\ \frac{1}{2} & t = 0 \\ 1 & 0 < t < L \\ \frac{1}{2} & t = L \\ 0 & t > L \end{cases}$$
`CStep`	continuous step function: $$u_{-1}(t) = \begin{cases} 1 & t > 0 \\ \frac{1}{2} & t = 0 \\ 0 & t < 0 \end{cases}$$
`Delta`	Dirac delta function $\delta(t)$ which has a value of zero for $t \neq 0$, a value of infinity at $t = 0$, and unit area: $$\int_{-\infty}^{\infty} \delta(t)\, dt = 1$$
`Dirichlet`	normalized Dirichlet discrete kernel: $$d[N, \omega] = \frac{\sin(N\omega/2)}{N \sin(\omega/2)}$$ Also called the `ASinc` function
`Impulse`	Kronecker delta function: $$\delta[n] = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases}$$
`Pulse`	discrete pulse: $$\sqcap_L[n] = \begin{cases} 0 & n < 0 \\ 1 & 0 \leq n \leq L - 1 \\ 0 & n \geq L \end{cases}$$
`Sinc`	$\mathrm{sinc}(t) \equiv \sin(t)/t$ so $\mathrm{sinc}(0) = 1$
`Step`	discrete step function: $$u[n] = \begin{cases} 1 & n \geq 0 \\ 0 & n < 0 \end{cases}$$
`Unit`	family of functions which includes `CStep` and `Delta`

Table 4.1: Signals Introduced by the Signal Processing Packages

Separable multidimensional signals are simply products of one-dimensional functions. The positive quadrant for the discrete-time variables n_1 and n_2 would be written as $u[n_1]u[n_2]$ which is rendered as `Step[n1] Step[n2]` in *Mathematica*. Non-separable multidimensional signals are easy to represent as well. For example, the line impulse through the origin running at a 45 degree angle in the continuous-time variables t_1 and t_2 would be written as $\delta(t_1 - t_2)$ which is rendered simply as `Delta[t1 - t2]` in *Mathematica*.

4.2 Defining Systems

The MDSPPs define the operators commonly used in signal processing but missing in *Mathematica*. Table 4.2 lists the new operators and their parameters. Parameters are side information such as shift factors, DFT lengths, and the variable(s) on which to operate. Operators also take arguments (i.e. signals):

$$operator \texttt{ [} parameter_1, parameter_2, \ldots \texttt{] [} signal_1, signal_2, \ldots \texttt{]}$$

Cascaded systems are formed by nesting operators. For example, with `Times` as the built-in product operator which takes no parameters, the expression

```
Times[ Exp[-2 I Pi t / 10], Shift[10,t][x[t]] ]
```

represents the modulation by $\exp(-2j\pi t/10)$ of $x(t-10)$ which is in turn *represented* by the `Shift` operator (note that a space between terms denotes multiplication). All operators in Table 4.2 *represent* operations in the sense that they defer evaluation of operations until `TheFunction` or `N` is applied to them. That is, when an operator from Table 4.2 is applied, no evaluation takes place— instead, the resulting function is stored symbolically, until it becomes convenient to compute it explicitly.

For one-dimensional operators, a variable is a symbol, like `t`. Arguments other than variables can be numbers, symbols or formulas. To indicate that the operator is multidimensional, we use a list of variables such as `{t1, t2, t3}` instead of a single

`Aliasby[sc, w]`	aliases a continuous function of `w` giving it a period of 2π / `sc` and divides by `sc`
`CircularShift[n0, N, n]`	shifts a sequence by `(n0 + n) mod N` in `n`
`CConvolve[t]`	continuous convolution in `t`
`Convolve[n]`	discrete convolution in `n`
`DFT[L, n, k]`	the `L`-sample (in 1-D) and the $\lvert\det$`L`$\rvert$-sample (in m-D) discrete Fourier transform of a function in `n` to a function in `k`
`DTFT[n, w]`	the discrete-time Fourier transform of a sequence in `n` to a continuous periodic function in `w`
`Difference[i, n]`	the ith backward difference in `n`
`Downsample[m, n]`	with the downsampling factor f equal to `m` in 1-D or $\lvert\det$`m`$\rvert$ in m-D, operator keeps the first sample in every block of f samples; the sampling rate decreases by a factor of f
`FIR[n, {h0, h1, ...}]`	all-zero digital filter with finite impulse response; in 1-D, the weights (taps) are `h0, h1, ...` and in m-D, the weights become a m-D volume of weights
`FT[t, w]`	continuous Fourier transform of a function of `t` into a function of `w`
`IIR[n, {a0, a1, ...}]`	all-pole digital filter with infinite impulse response; in 1-D, the feedback coefficients are `-a1`, ..., and in m-D, `{a0, a1, ...}` becomes a m-D volume of coefficients
`Interleave[n]`	interleaves (combines) samples from each input function of `n` into one function
`InvDFT[L, k, n]`	inverse discrete Fourier transform (see `DFT`)
`InvDTFT[w, n]`	inverse discrete-time Fourier transform (see `DTFT`)
`InvFT[w, t]`	inverse continuous Fourier transform (see `FT`)
`InvL[s, t]`	inverse Laplace transform (see `L`)
`InvZ[z, n]`	inverse z-transform (see `Z`)
`L[t, s]`	Laplace transform of a function of `t`
`Periodic[p, v]`	argument is made periodic with period `p` with respect to variable(s) `v`; in m-D, `p` is an integer matrix
`Rev[v]`	reverse the function of variable `v` (flip the `v` axis)
`ScaleAxis[sc, w]`	scale the `w` axis (variable) by `sc`
`Shift[v0, v]`	shift function of `v` by `v0` points
`Summation[i, ib, ie, inc]`	summation operator: `i` = `ib` to `ie` step `inc`
`Upsample[l, n]`	with the upsampling factor f equal to `l` in 1-D or $\lvert\det$`l`$\rvert$ in m-D, operator inserts $f-1$ samples after each input sample; sampling rate increases by a factor of f
`Z[n, z]`	z-transform of a function of `n`

Table 4.2: New Operators (and Their Parameters) in the MDSPPs

variable such as `t`. The length of the variable list indicates the dimensionality of the operator.

In multiple dimensions, the first parameter for the upsampling, downsampling, and periodic operators becomes a square matrix, whereas all other parameters become lists of expressions, one expression per dimension. For example, the downsample operator requires a downsampling constant `m` and a symbolic variable `n`. In one-dimensional downsampling, `m` is the integer downsampling factor and `n` is a symbolic index, but for the multidimensional case, `m` is the integer downsampling matrix and `n` is a list of symbolic indices. Regardless of the dimensionality, the form of the operator is `Downsample[m, n]`, and its syntax is `Downsample[m, n][f]`, where f is a discrete function in `n`.

4.3 Plotting Signals and Systems

All of the continuous-time signals introduced by the MDSPPs, except for the Dirac delta function, can be graphed by *Mathematica*'s basic plotting command `Plot`. `Plot`, however, will experience problems if the signal contains any of the new operators introduced by the MDSPPs, or if the signal contains Dirac delta functions, or if the signal is complex-valued. Thus, we introduce a new function called `SignalPlot`. `SignalPlot` will first convert the expression into functions which can be plotted and then graphs the real and imaginary parts of the function in solid and dashed lines, respectively. This plotting convention for analog signals follows Bracewell [74]. `SignalPlot` also plots Dirac delta functions as upward- or downward-pointing arrows. The height of the arrows is either the area of the delta function (when `$DeltaFunctionScaling` is set to `Scaled`) or the same height which is determined from the values of the rest of the plot (when `$DeltaFunctionScaling` is set to `None`). For discrete-time signals, we have introduced the function `SequencePlot` that only plots real-valued functions of an integral index variable.

Both new plotting routines can plot one-dimensional and two-dimensional signals. For two-dimensional sequences, `SequencePlot` generates three-dimensional graphics initially sampled at discrete indices. Sometimes, a better way to visualize a two-dimensional sequence is to view it at an infinite height above the plane of the two independent variables, known as a density plot. A density plot, for example, shows clearly the sampling pattern after multidimensional upsampling. For example, Figure 4.1 shows how to use *Mathematica*'s `DensityPlot` command with our upsampling operator to illustrate the effect of upsampling a separable sequence by the non-rectangular upsampling matrix $\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$, which has an upsampling factor of $1 \cdot 1 - (-1) \cdot 1 = 2$. First, the upsampling changes the basis of the sampling lattice to a non-rectangular lattice so that the function becomes non-separable. Second, the upsampling inserts zeros in a checkerboard pattern as seen by the black tiles which represent an amplitude of zero. (This checkerboard pattern corresponds to interleaving in digital video signals, and the resampling matrix involved belongs to the family of *quincunx* resampling matrices.) In Figure 4.1, the white tiles represent an amplitude of two.

4.4 Continuous and Discrete Convolution

Convolution is a key concept in linear shift-invariant systems as it describes filtering in the time domain. Continuous convolution of two functions, $f(t)$ and $g(t)$, is defined as [5]

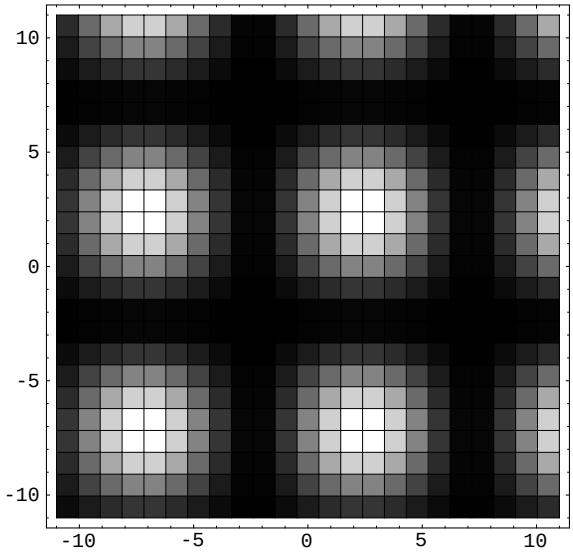
$$f(t) \star g(t) = \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau$$

In discrete time, the integral is replaced by summation [75]

$$f[n] \star g[n] = \sum_{m=-\infty}^{\infty} f[m]\, g[n-m]$$

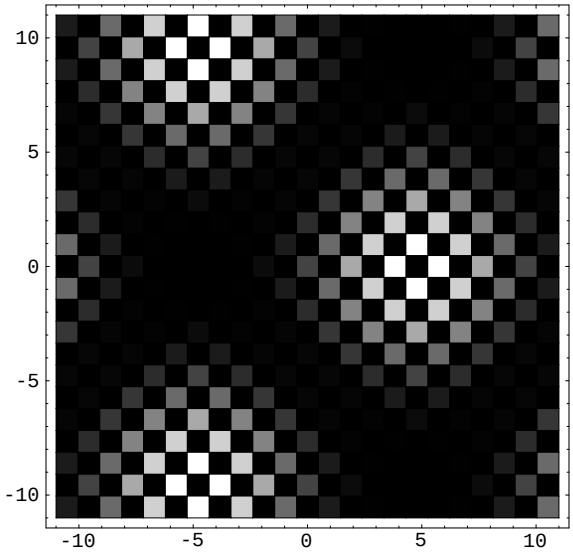
The graphical approach to computing one-dimensional convolutions flips g in time and slides it across f. The answer is obtained by performing the integration or summation over those intervals where the functions overlap.

```
sin2D = ( Sin[n1 Pi / 5] + 1 ) *
        ( Sin[n2 Pi / 5] + 1 );

DensityPlot[
     sin2D,
     {n1, -11, 11}, {n2, -11, 11},
     PlotPoints -> {23, 23},
     PlotRange -> All ]
```




```
-DensityGraphics-
```

```
upMatrix = {{1, 1}, {-1, 1}};
upSignal =
     Upsample[upMatrix, {n1,n2}][sin2D];
DensityPlot[
     Evaluate[TheFunction[upSignal]],
     {n1, -11, 11}, {n2, -11, 11},
     PlotPoints -> {23, 23},
     PlotRange -> All ]
```




```
-DensityGraphics-
```

Figure 4.1: Visualizing Two-Dimensional Sequences as Density Plots
Top views of a separable function (left) and the upsampling of the same function by a non-rectangular matrix (right).

The MDSPPs introduce `PiecewiseConvolution` which performs discrete convolution and convolution for piecewise continuous functions. `PiecewiseConvolution` accepts f and g as algebraic expressions or as lists of F-intervals. West and McClellan [26] define the F-interval as a list of three elements:

$$\{\textit{function},\ \textit{left-endpoint},\ \textit{right-endpoint}\}$$

F-intervals can be finite or infinite in extent, but when given numeric values, the right endpoint should always be greater than the left endpoint. The *function* can be any algebraic expression, e.g. `a t^2`. As a list of F-intervals, then, `t CPulse[10, t] + 5 Delta[t - 20]` becomes { {`t, 0, 10`}, {`Area[5], 20, 20`} }.

Internally, `PiecewiseConvolution` uses the following algorithm to compute convolutions [26]:

1. convert f and g to lists of F-intervals,
2. convolve each F-interval in f with each F-interval in g, and
3. simplify the result by
   (a) reordering (sorting) the F-intervals and
   (b) merging overlapping F-intervals (when possible).

West [26] bases the computation of the convolution of two F-intervals on the "Square Matrix Rule" [76] which says that convolving two finite extent intervals produces three finite extent intervals. We have generalized the rule to include situations where one or more of the endpoints is infinite as shown in Table 4.3. Since each left endpoint in an interval can either be $-\infty$ or a number and each right endpoint in an interval can either be a number or ∞, convolving two F-intervals yields $2 \cdot 2 \cdot 2 \cdot 2 = 16$ possible combinations of intervals. All of the combinations yield one, two, or three intervals. We have enumerated all possible combinations of intervals in validating Table 4.3.

Interval	Value	Valid
$t \in (-\infty, l_f + l_g)$	0	always
$t \in [l_f + l_g, l_f + u_g)$	$\int_{l_f}^{t-l_g} f(\tau)\, g(t-\tau) d\tau$	if $l_f \neq -\infty$ and $l_g \neq -\infty$
$t \in [l_f + u_g, u_f + l_g)$	$\int_{t-u_g}^{t-l_g} f(\tau)\, g(t-\tau) d\tau$	if $l_g \neq -\infty$ and $u_g \neq \infty$
$t \in [u_f + l_g, u_f + u_g)$	$\int_{t-u_g}^{u_f} f(\tau)\, g(t-\tau) d\tau$	if $u_f \neq \infty$ and $u_g \neq \infty$
$t \in [u_f + u_g, \infty)$	0	always

(a) Formulas for continuous-time convolution

Interval	Value	Valid
$n \in (-\infty, l_f + l_g - 1]$	0	always
$n \in [l_f + l_g, l_f + u_g - 1]$	$\sum_{m=l_f}^{n-l_g} f[m]\, g[n-m]$	if $l_f \neq -\infty$ and $l_g \neq -\infty$
$n \in [l_f + u_g, u_f + l_g]$	$\sum_{m=n-u_g}^{n-l_g} f[m]\, g[n-m]$	if $l_g \neq -\infty$ and $u_g \neq \infty$
$n \in [u_f + l_g + 1, u_f + u_g]$	$\sum_{m=n-u_g}^{u_f} f[m]\, g[n-m]$	if $u_f \neq \infty$ and $u_g \neq \infty$
$n \in [u_f + u_g + 1, \infty)$	0	always

(b) Formulas for discrete-time convolution

I. if $l_f = -\infty$ and $u_f = \infty$ OR $l_g = -\infty$ and $u_g = \infty$, then the convolution extends for all time and is computed by the definition.

II. if $l_f = -\infty$ and $u_f \neq \infty$ and $l_g \neq -\infty$ and $u_g = \infty$, then the convolution extends for all time with two intervals of interest for the output variable v:

   A. for $v \in (-\infty, u_f)$, apply definition with upper limit being v, and

   B. for $v \in [u_f, \infty)$, apply definition with upper limit being u_f.

III. if $l_f \neq -\infty$ and $u_f = \infty$ and $l_g = -\infty$ and $u_g \neq \infty$, then apply II above after switching the two F-intervals.

Table 4.3: Modification to the Square Matrix Rule for Convolution
Convolving one F-interval $\{f,\ l_f,\ u_f\}$ with another F-interval $\{g,\ l_g,\ u_g\}$ generates one, two, or three intervals with non-zero functions. The tables assume that the first F-interval is the longer one, i.e. $u_f - l_f > u_g - l_g$. The three special cases are enumerated above.

Even though `PiecewiseConvolution` does not use the flip-and-slide approach to calculate outputs, the routine does have the ability to demonstrate the flip-and-slide approach by means of animation, e.g. convolving two pulse functions in t:

```
PiecewiseConvolution[ CPulse[1,t], CPulse[1,t], t, Dialogue -> All ]
```

Enabling the `Dialogue` option triggers the generation of the animation sequence: `False` for no animation, `True` for the flip-and-slide of the first function across the second, and `All` for the flip-and-slide of the first function across the second as well as the corresponding overlapping intervals for each flip-and-slide frame. The `Dialogue` option is used in many other routines in the MDSPPs, as is demonstrated in Chapter 5.

4.5 Summary

Mathematica possesses some inherent signal processing capabilities. It can compute the numerical (multidimensional) Fourier transform of a (multidimensional) sequence of data. It defines many functions such as trigonometric, exponential, logarithmic, and Bessel functions commonly used in an algebraic representation of signals. However, basic signals such as step and impulse functions and basic systems such as shifters and filters are missing in the core of *Mathematica*. By extending *Mathematica*, the MDSPPs define many of the missing signals and systems. This chapter also introduces some of the plotting and symbolic abilities of the MDSPPs. Examples of plotting in the "time" domain include signal plots, sequence plots, and density plots (for the visualization of resampling operations). The lone symbolic ability introduced by this chapter is linear convolution which is implemented for both the discrete and continuous domains. Chapter 5 describes the MDSPPs in more detail.

CHAPTER 5

Analysis of Multidimensional Signals and Systems

Chapter 2 introduces SPLICE and its descendants which can deduce certain properties of one-dimensional discrete-time signals generated by various components of a system. The multidimensional signal processing packages (MDSPPs) extend the ability of SPLICE to analyze signal properties to multiple dimensions. As discussed in Section 5.1, several new important properties have also been added.

Besides generalizing the analysis of signal properties to multiple dimensions, the MDSPPs define a suite of analysis tools for multidimensional discrete-time *and* continuous-time signals based on linear transform theory. For continuous-time signals, the MDSPPs implement the multidimensional Laplace and Fourier transforms. For discrete-time signals, the MDSPPs implement the multidimensional z, discrete-time Fourier, and discrete Fourier transforms. The linear transform routines are *symbolic* because they map algebraic expressions between the "time" and various "frequency" domains (see Section 5.2).

Analysis of separable multidimensional systems reduces to analysis of each dimension separately. The one-dimensional capabilities of the multidimensional z and Laplace transform routines are useful in solving linear constant coefficient difference and differential equations, e.g. those describing passive networks (see Section 5.3.1). Coupling the symbolic linear transforms with graphical representations of signals enables new routines to automate the analysis of one-dimensional signals in the time, generalized frequency (z or Laplace), and Fourier frequency domains (see Section 5.3.2).

The linear transform routines are extensible in that users can provide their own transform pairs. Specifying transform pairs for input signals enables the transform routines to generate the input-output relationships of multirate systems (see Section 5.4).

The forward z and Laplace transform routines support multidimensional multilateral signals, so they must track the region of convergence (ROC). The ROC can be used to determine algebraic conditions for the stability of multidimensional systems (see Section 5.5.1). After assigning values to free parameters, stability can be determined graphically for two-dimensional systems (see Section 5.5.2).

The multidimensional capabilities of the transform routines assist in the automation of signal analysis and the derivation of transform properties. Automated two-dimensional signal analysis (Section 5.6.1) is an extension of the general signal analyzers described earlier. Deriving properties of linear transforms (including tracking the changes in the region of convergence) is now possible in higher dimensions (Section 5.6.3).

5.1 Extending Signal Properties in E-SPLICE and ADE

Myers [7, 8, 9] identified a set of properties for one-dimensional signals as shown earlier in Table 2.1. Each of these properties can be generalized to the multidimensional case. The `Period` property is now described by a non-singular integer periodicity matrix (which is diagonal for separable signals). We have made `Start-BW` and `End-BW` more descriptive by renaming them to `StartBandwidth` and `EndBandwidth`, respectively. The properties of `Start`, `End`, `StartBandwidth` and `EndBandwidth` become sets of coordinates. Similarly, `Support` and `Bandwidth` become lists with one element per dimension. Together, these two slots (pieces of data) define the smallest non-zero rectangular region of support containing the "time" and "frequency"

domains, respectively. We add the methods (procedures) `InDomainFunction` and `InBandwidthFunction` to define the shape of the region of support. For example, a circular region of support for the general multidimensional case would be written as

```
Function[coordinates, Apply[Plus, coordinates^2] < r^2]
```

which computes $x_1^2 + x_2^2 + \cdots < r^2$ where $(x_1, x_2, \ldots)$ is the list of `coordinates`. The methods `InDomainQ` and `InBandwidthQ` check first if the passed set of coordinates is in the rectangular region of support and then call `InDomainFunction` or `InBandwidthFunction` to determine if the set of coordinates is really in the region of support. Many types of symmetry can exist for different combinations of the signal variables, so the properties `Symmetry` and `CenterOfSymmetry` become lists of the types of and centers of symmetry, respectively. Defining the types of symmmetry and their centers, however, in multiple dimensions becomes very difficult. The `Real-or-Complex` property has been replaced by `DataType` which can take the value of `Real`, `Complex`, `Integer`, etc. The MDSPPs add the property `Variables` to keep track of the "time" domain variables. Table 5.1 lists the multidimensional signal properties recognized by the MDSPPs.

5.2 Linear Transforms

Many signal processing algorithms are ultimately based on linear transforms— z, discrete-time Fourier, discrete Fourier for discrete-time signals and Laplace and Fourier transforms for continuous-time signals. The multidimensional signal processing packages, which are introduced in Chapter 4, implement these transforms in their most general multidimensional multi-sided symbolic forms by applying a sequence of transformation rules. Each routine first checks special multidimensional rules and then applies the one-dimensional rule base to each dimension of the signal. Even though we use the fact that the linear transform kernels are separable, we do not make any

Signal Property	Meaning
`Bandwidth`	non-zero rectangular extent of frequency domain
`CenterOfSymmetry`	center of symmetry in the time domain
`DataType`	value is `Real`, `Complex`, `Integer`, ...
`End`	end of non-zero rectangular extent in the time domain
`EndBandwidth`	end of non-zero rectangular extent of bandwidth
`InBandwidthFunction`*	function defining the shape of the frequency domain
`InBandwidthQ`*	checks to see if the coordinates are in the region of support for the time domain
`InDomainFunction`*	function defining the shape of the time domain
`InDomainQ`*	checks to see if the coordinates are in the region of support for the frequency domain
`Period`	(multidimensional) period
`Start`	start of non-zero rectangular extent in the time domain
`StartBandwidth`	start of non-zero rectangular extent of bandwidth
`Support`	non-zero rectangular extent in the time domain
`Symmetry`	multi-valued parameter describing time-domain symmetry; possible values are: `Symmetric`, `AntiSymmetric`, `ConjugateSymmetric`, and `ConjugateAntiSymmetric`
`Variables`*	symbols used as time domain variables

* new multidimensional property (unnecessary in one dimension)

Table 5.1: Signal Properties Supported by the Signal Processing Packages

assumptions about the signal being transformed.

The rules in each one-dimensional rule base are applied sequentially. At each iteration, the first rule that applies to the expression being transformed is invoked thereby rewriting the expression. The process repeats until the expression no longer changes. Since the rules are applied in sequential order, the order of the rules is very important. We have encoded the rules in the order that humans tend to use them: transform pairs, transform properties (e.g. additivity), transforms of operators (e.g. shift), and transform strategies (e.g. partial fraction decomposition). Figure 5.1 diagrams the general structure of the one-dimensional rule bases.

All of the inverse transform routines return a final algebraic expression, whereas the forward transform routines return a data structure. The forward z and Laplace transforms track and return the region of convergence (ROC). The ROC is necessary to guarantee uniqueness because the z and Laplace transforms are multilateral. All of the linear transforms can display the intermediate calculations so that designers can check the computation. Users can also specify their own transform pairs. This feature is useful in generating transfer functions for systems (as shown in Section 5.4) and deriving properties of transforms (as shown in Section 5.6.3).

5.3 Analysis of One-Dimensional Systems Using Transforms

5.3.1 Solving Difference and Differential Equations

The linear differential equation solver `LSolve` uses the unilateral Laplace transform to solve one-dimensional linear constant coefficient differential equations. Figure 5.2 shows the steps taken by `LSolve` in solving for $y(t)$ in the equation

$$y''(t) + \frac{3}{2}y'(t) + \frac{1}{2}y(t) = e^{at}u_{-1}(t)$$

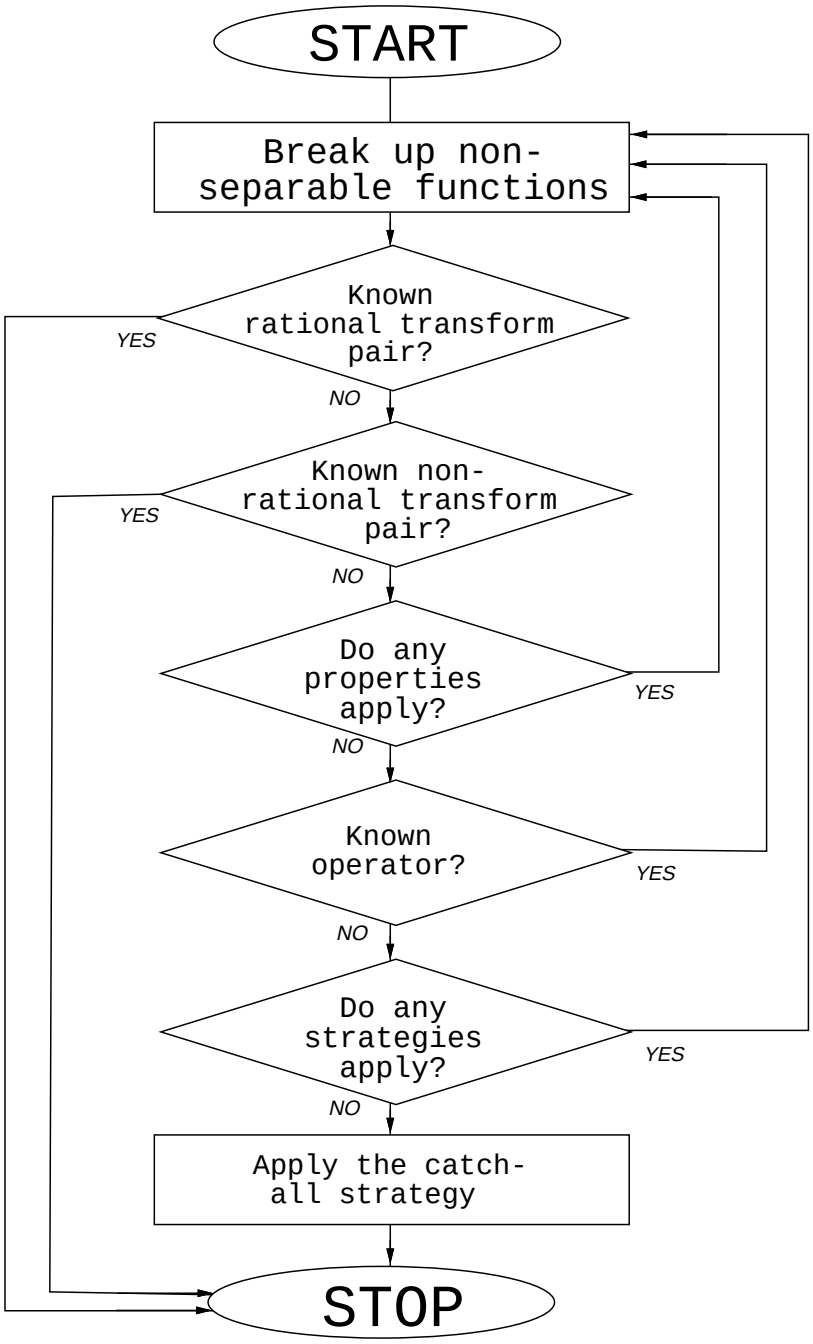


Figure 5.1: Structure of the One-Dimensional Transform Rule Bases
One-dimensional transform rule bases form the backbone of the more general multidimensional, multi-sided transforms. Each rule base calls a one-dimensional rule base once for each dimension of the transform.

```
In[17]:=
LSolve[ y''[t] + 3/2 y'[t] + 1/2 y[t] ==
        Exp[a t] CStep[t], y[t], y[0] -> 4,
        y'[0] -> 4, Dialogue -> All ]

Solving for y[t] in the differential equation

  y[t]   3 y'[t]              a t
  ---- + ------- + y''[t] = E    CStep[t]
   2        2

  subject to the initial conditions

  {y[0] -> 4, y'[0] -> 4}


The Laplace transform of the left side is:

                1   3 s    2    3 y[0]
  L { y[t] } (- + --- + s ) - ------ -
                2    2           2

   s y[0] - y'[0]

( In the general case, the unilateral Laplace
  transform of the nth derivative of y[t] is:

     (n)                        n
  L {y   [t] } = L { y[t] } s  +

     -1 + n              -2 + n
    s       y[t0]       s       y'[t0]
  -(-------------) + -(--------------) + ... +
        s t0                s t0
       E                   E

     (-1 + n)
    y        [t0]
  -(-------------)
        s t0
       E

  where t=t0 is the initial condition. )


The Laplace transform of the right side is:

    1
  -----
  s - a

Solving for the unknown transform yields

     1     3 y[0]
  ----- + ------ + s y[0] + y'[0]
  s - a      2
  -------------------------------
            1   3 s    2
            - + --- + s
            2    2

which becomes

                1
  10 + 4 s + -----
             s - a
  ----------------
    1   3 s    2
    - + --- + s
    2    2

Inverse transforming this gives y[t]:

Out[17]=
  (2 (-5 - 16 a - 12 a^2 + 6 E^(t/2) + 22 a E^(t/2) +
       16 a^2 E^(t/2) + E^t + a t) CStep[t]) /
    ((1 + 3 a + 2 a^2) E^t)

In[18]:=
  yoft = % /. ( CStep[t] -> 1 )

Out[18]=
  (2 (-5 - 16 a - 12 a^2 + 6 E^(t/2) + 22 a E^(t/2) +
       16 a^2 E^(t/2) + E^t + a t)) /
    ((1 + 3 a + 2 a^2) E^t)

In[19]:=
  Simplify[ yoft /. t -> 0 ]

Out[19]=
  4

In[20]:=
  Simplify[ D[ yoft, t ] /. t -> 0 ]

Out[20]=
  4
```

Figure 5.2: Interaction with the Differential Equation Solver

subject to the initial conditions $y(0^+) = 4$ and $y'(0^+) = 4$. Because full justification is enabled by the option `Dialogue -> All`, the solver describes the intermediate calculations. The user has access to the answer as well as how to obtain it. Once the answer has been obtained, the user can check the answer. Command 18 defines the variable `yoft` as $y(t)$ for $t > 0$; since $u_{-1}(t) = 1$ for $t > 0$, the continuous-time step function is replaced by 1. Command 19 verifies that $y(0+)$ is indeed 4 and the last command checks the value of $y'(0+)$.

Like `LSolve`, `ZSolve` uses the unilateral z-transform to solve one-dimensional initial value linear constant coefficient difference equations. Figure 5.3 shows the steps taken by the difference equation solver to compute the closed-form formula for the well-known Fibonacci sequence:

$$y_n - y_{n-1} - y_{n-2} = 0 \qquad y_0 = 0 \qquad y_1 = 0$$

The solver computes the non-recursive formula for y_n in exact precision. Even though the solution contains irrational numbers, the formula yields integers for the Fibonacci numbers because arithmetic is performed in exact precision. The first 11 values of the sequence are computed by the `Table` command in Figure 5.3.

5.3.2 Generalized Signal Analysis

The MDSPPs support new formats for plotting signals, sequences, and their transforms. New routines plot 1-D and 2-D signals and systems in the time domain as well as their pole-zero diagrams and frequency responses. They can generate several different styles of frequency plots, i.e. whether the frequency domain scale is linear or logarithmic and whether the magnitude plot has a linear or decibel range. They also graph root loci for one varying parameter.

Two routines, `ASPAnalyze` and `DSPAnalyze`, combine these plotting abilities with the transform capabilities to perform a complete analysis of 1-D and 2-D signals. These analyzers display textual and graphical information. The textual dialogue

```
fib[n_] =
     ZSolve[ y[n] - y[n-1] - y[n-2] == 0,
             y[n], y[0] -> 0, y[1] -> 1,
             Dialogue -> All ]
```

Solving for y[n] in the difference equation
  -y[-2 + n] - y[-1 + n] + y[n] = 0
  such that n > 1 and subject to the
  initial conditions
  {y[0] -> 0, y[1] -> 1}

The first step is to substitute the initial
  conditions into the difference equation
  for n from 0 to 1. This will give rise
  to 2 impulse function(s) which will be
  added to the right-hand side of the
  difference equation. Then, the unilateral
  z-transform will be used to solve the
  difference equation in the index n.

Solving for y[n] in the difference equation
  augmented by the initial conditions:
  -y[-2 + n] - y[-1 + n] + y[n] =
    Impulse[-1 + n]

The z-transform of the left side is:

$$\left(1 - z^{-2} - \frac{1}{z}\right) Z\{ y[n] \}$$

The z-transform of the right side is:

$$\frac{1}{z}$$

Solving for the unknown z-transform yields

$$\frac{1}{\left(1 - z^{-2} - \frac{1}{z}\right) z}$$

Inverse transforming this gives y[n]:

$$\frac{-2\ 2^{-1+n} \left(\frac{1}{-1 - \text{Sqrt}[5]}\right)^{-1+n} \text{Step}[-1+n]}{\text{Sqrt}[5]\ (-1 - \text{Sqrt}[5])} + \frac{2\ 2^{-1+n} \left(\frac{1}{-1 + \text{Sqrt}[5]}\right)^{-1+n} \text{Step}[-1+n]}{\text{Sqrt}[5]\ (-1 + \text{Sqrt}[5])}$$

```
Table[ Simplify[fib[i]], {i, 0, 10} ]
{0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55}
```

Figure 5.3: Solving the Fibonacci Difference Equation

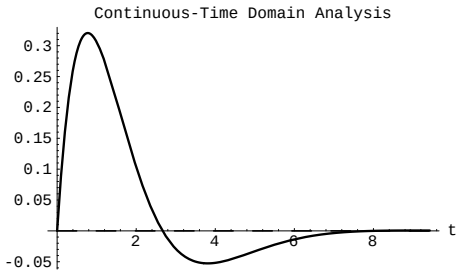
includes the Fourier and the generalized transform (z or Laplace) as well as the algebraic conditions for stability in terms of the free parameters. Graphics information includes the time-domain plot, the pole-zero diagram, and the frequency response. The output of the signal analysis routines is meant to be self-explanatory at the level of a student studying linear systems theory. For example, Figure 5.4 shows the general analysis of the impulse response of an analog filter. Also, Figure 5.5 shows general analysis of the upsampling by 3 of a discrete-time (i.e. sampled) sinc function.

5.4 Analysis of Multirate Systems Using Transforms

One way to understand the behavior of a system is to derive its input-output relationship in a transform domain. The input-output relationship is simply a transfer function for linear shift-invariant (LSI) systems, but it becomes a transfer function plus aliasing terms for linear periodically time-varying (LPTV) systems [49]. When deriving input-output relationships on paper, engineers implicitly use transform pairs such as $x[n] \longleftrightarrow X(z)$ to represent the z-transform of the input signal $x[n]$. For LSI and LPTV systems, the linear transform routines can derive input-output relationships if all of the unknown quantities (input signals, unspecified filters, and so forth) are assigned transform pairs. This ability applies to one-dimensional and multidimensional, as well as continuous-time and discrete-time, systems. Discrete-time LPTV systems are also known as multirate systems.

Figure 5.6 shows the derivation of the input-output relationship for a two-channel non-uniform one-dimensional filter bank. The block diagram of the filter bank is shown in Figure 5.6(a). In the filter bank, the upper channel contains the frequency band $(-\frac{2}{3}\pi, \frac{2}{3}\pi)$ of the input signal, whereas the lower channel contains the frequency band $(-\pi, -\frac{2}{3}\pi) \cup (\frac{2}{3}\pi, \pi)$. Both channels are resampled at their Nyquist rates. Figure 5.6(b) gives the definition of the filter bank structure using the notation of the signal

```
In[2]:=
ASPAnalyze[ t Exp[-a t] Cos[3 Pi t / 16] CStep[t],
            t, 0, 3 Pi, a -> 1 ]

For plotting only, these symbols will
will be assigned default values:  {a -> 1}

Real part of the function is shown as solid lines.
Imaginary part of the function is shown as dashed
  lines.
```



$$\frac{3\,\text{Pi}\,t}{t\,\text{Cos}[\text{------}]\,\text{CStep}[t]}{16}{\text{a t}}{\text{E}}$$

has the following Laplace transform:

$$\frac{256\,(256\,a^2 - 9\,\text{Pi}^2 + 512\,a\,s + 256\,s^2)}{(256\,a^2 + 9\,\text{Pi}^2 + 512\,a\,s + 256\,s^2)^2}$$

The region of convergence is:

-Re[a] < Re(s) < Infinity

The system is stable if -Re[a] < 0

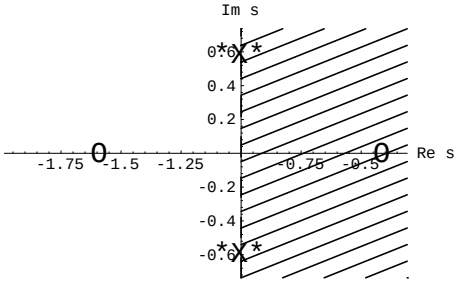
The default values are now being considered.

The zeroes are: {-1.58905, -0.410951}

The poles are: {-1. + 0.589049 I,

-1. - 0.589049 I, -1. + 0.589049 I,

-1. - 0.589049 I}

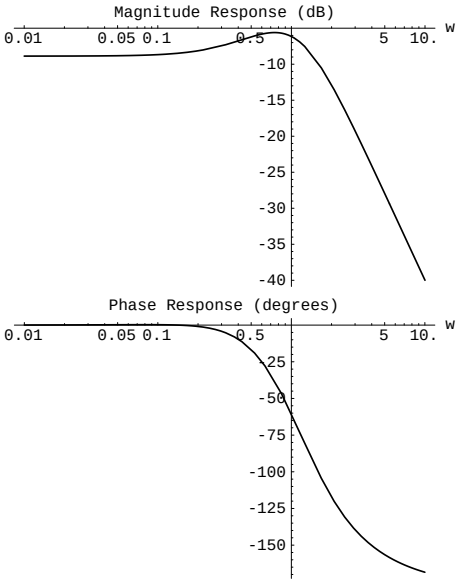


Since the signal is stable, the frequency response can be computed directly from the Laplace transform.

$$\frac{3\,\text{Pi}\,t}{t\,\text{Cos}[\text{------}]\,\text{CStep}[t]}{16}{\text{a t}}{\text{E}}$$

has the following frequency response:

$$\frac{256\,(256\,a^2 - 9\,\text{Pi}^2 + 512\,\text{I}\,a\,w - 256\,w^2)}{(256\,a^2 + 9\,\text{Pi}^2 + 512\,\text{I}\,a\,w - 256\,w^2)^2}$$



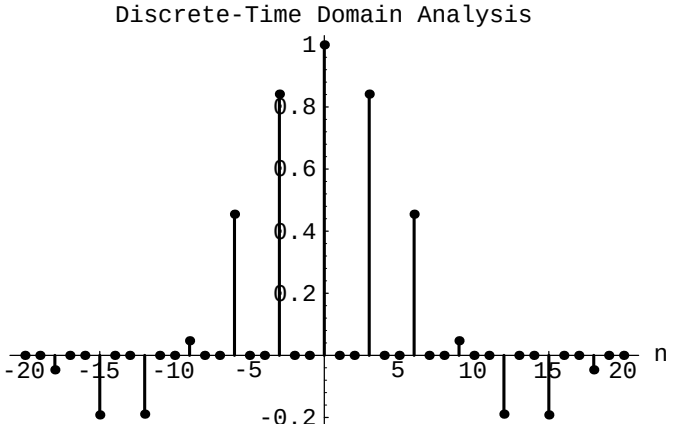

```
Out[2]=
LTransData[
   256 (256 a^2 - 9 Pi^2 + 512 a s + 256 s^2)
   ------------------------------------------,
    (256 a^2 + 9 Pi^2 + 512 a s + 256 s^2)^2
  Rminus[-Re[a]], Rplus[Infinity],
  LVariables[s]]
```

The signal analyzer plots the impulse response of the of the analog filter over $t \in (0, 3\pi)$ when the parameter `a` takes value of 1. The analyzer finds the Laplace transform and displays the pole-zero diagram (with the ROC shaded). It also prints the formula describing the frequency response and plots the magnitude and phase of the frequency response.

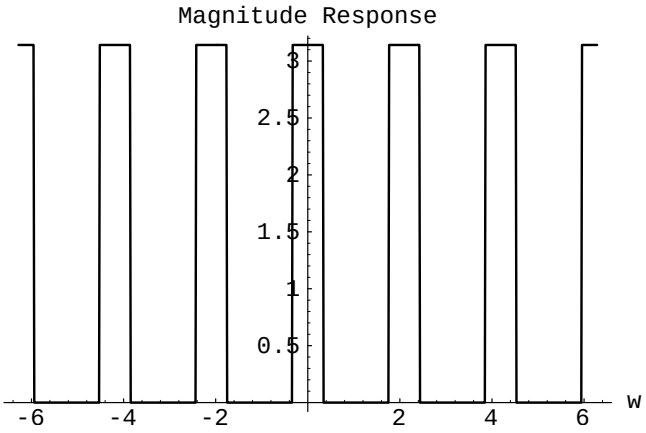
Figure 5.4: One-Dimensional Analog Signal Analysis

In[2]:=

```
DSPAnalyze[ Upsample[3,n][ Sinc[n] ], n, -20, 20 ]
```



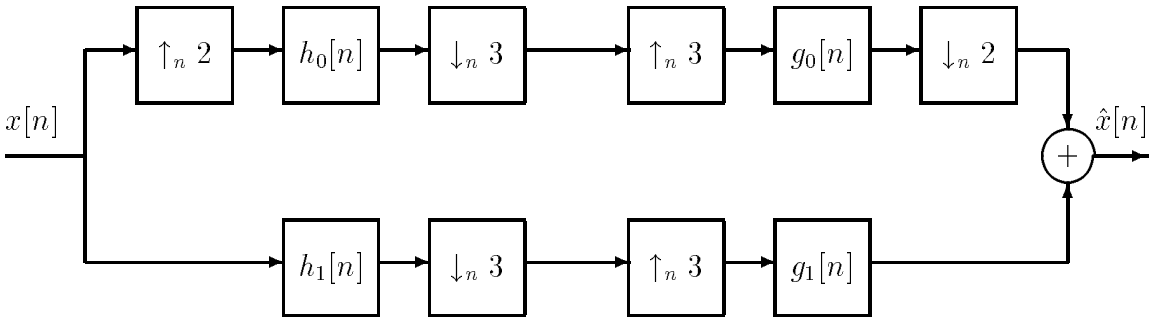

```
Transform::incomplete:
   The rule base could not compute the forward
   z-transform of Sinc[n] with respect to n.

Upsample   [Sinc[n]]
        3,n

has the following frequency response:

                                          1
Pi Periodic           [CPulse   [- + w]]
           2 Pi              2/3 3
           ----  in  w
            3
```



Out[2]=

```
-Incomplete z-Transform-
```

Figure 5.5: One-Dimensional Discrete-Time Analysis of a Multirate Signal
The z-transform does not exist for this case, so only the time-domain signal and its Fourier transform are displayed.



(a) Flow graph of the filter bank

```
upperchannel =
  Downsample[2,n][
    Convolve[n][g0[n], Upsample[3,n][
                         Downsample[3,n][
                           Convolve[n][h0[n],
                                       Upsample[2,n][x[n]]]]]]]
lowerchannel =
  Convolve[n][g1[n], Upsample[3,n][
                       Downsample[3,n][ Convolve[n][h1[n], x[n]]]]]
```

(b) Representation of the filter bank in the new environment

```
Xhat[z] = ZTransform[ lowerchannel + upperchannel, n, z,
                      TransformLookup -> { x[n] :> X[z],
                           g0[n] :> G0[z], g1[n] :> G1[z],
                           h0[n] :> H0[z], h1[n] :> H1[z] } ]
```

(c) *Mathematica* code to generate the input-output relationship

```
 TeXForm[ Collect[ SPSimplify[ 6 TheFunction[ Xhat[z] ] ],
                   { X[z], X[Exp[2 I Pi/3] z], X[Exp[4 I Pi/3] z] } ] ]
```

$$\hat{X}(z) = \tfrac{1}{6}\big(G_0(-\sqrt{z})H_0(-\sqrt{z}) + G_0(\sqrt{z})H_0(\sqrt{z}) + 2\,G_1(z)H_1(z)\big)\,X(z) +$$
$$\tfrac{1}{6}\Big(G_0(-\sqrt{z})H_0(e^{\frac{i}{3}\pi}\sqrt{z}) + G_0(\sqrt{z})H_0(e^{\frac{4i}{3}\pi}\sqrt{z}) + 2\,G_1(z)H_1(e^{\frac{2i}{3}\pi}z)\Big)\,X(e^{\frac{2i}{3}\pi}z) +$$
$$\tfrac{1}{6}\Big(G_0(\sqrt{z})H_0(e^{\frac{2i}{3}\pi}\sqrt{z}) + G_0(-\sqrt{z})H_0(e^{\frac{5i}{3}\pi}\sqrt{z}) + 2\,G_1(z)H_1(e^{\frac{4i}{3}\pi}z)\Big)\,X(e^{\frac{4i}{3}\pi}z)$$

(d) Input-output relationship generated by the environment

Figure 5.6: Deriving Input-Output Relationship for a Non-Uniform Filter Bank

processing packages. Figure 5.6(c) uses the forward z-transform routine to generate the relationship between $X(z)$ and $\hat{X}(z)$ in the z domain. Figure 5.6(d) converts the algebraic input-output relationship to its TEX form using the built-in abilities of *Mathematica*'s `TeXForm` command. In order to make the formula look better in TEX, we wrote the *Mathematica* command to multiply $\hat{X}(z)$ by six, then collect terms, and finally generate the equivalent TEX code. By hand, we added the $\frac{1}{6}$ factors to each term.

5.5 Multidimensional Stability Analysis

Multidimensional stability analysis is difficult to describe analytically. For some classes of multidimensional systems, the conditions for stability can be derived by examining the region of convergence (ROC) of their generalized frequency transforms (Section 5.5.1). Once values have been assigned to free parameters, stability becomes much easier to analyze by visualization (Section 5.5.2).

5.5.1 Symbolic Analysis of Stability

Knowing the ROC for a signal leads to a set of stability conditions involving the free parameters of the signal. A discrete-time signal is stable if the ROC of its z-transform $R_- < |z| < R_+$ contains the unit circle, i.e. $R_- < 1 < R_+$. A continuous-time signal is stable if the ROC of its Laplace transform $R_- < \Re e(s) < R_+$ contains the imaginary axis, i.e. $R_- < 0 < R_+$. The `Stable` command uses these simple comparisons to convert ROC information to a set of stability conditions.

These comparisons generalize to multiple dimensions. Figure 5.7 shows an example of the stability analysis of a two-dimensional non-separable signal taken from [53]. In multiple dimensions, non-separable signals have non-separable regions of convergence, which implies that one or more of the z-transform variables will appear in the ROC. When checking for stability, the z-transform variables take values on the

```
In[36]:=
  ZTransform[ (n1 + n2)! a^n1 b^n2 Step[n1,n2] / (n1! n2!),
              {n1, n2}, {z1, z2} ]
Out[36]=
                   1                                          1
  ZTransData[-----------, Rminus[{Abs[a], Abs[b] Abs[------]}],
                 a    b                                       a
             1 - -- - --                                1 - --
                 z1   z2                                    z1

             Rplus[{Infinity, Infinity}], ZVariables[{z1, z2}]]

In[37]:= Stable[ % ]
Out[37]= Abs[a] < 1 && Abs[b] < 1 - Abs[a]

In[38]:= Assuming[All]
Out[38]=
                                1                 a
  Abs[a] < 1 && Abs[b] Abs[------] < 1 && 1 - -- != 0
                                  a               z1
                             1 - --
                                 z1
```

The above interaction demonstrates the stability analysis of a two-dimensional non-separable sequence in the variables `n1` and `n2` having the form

$$\frac{(n_1+n_2)!}{n_1!\,n_2!}\;a^{n_1}\;b^{n_2}\;u[n_1,n_2]$$

Since the function is not separable, neither is the region of convergence (ROC). `Stable` generates the inequality for checking if the unit bi-sphere ($|z_i| = 1 \forall i$) is in the ROC. If the inequality does not evaluate to true or false, `Stable` will apply a set of simplification rules to it with the side information $|z_i| = 1 \forall i$. The actual stability condition is $|a| + |b| < 1$ [53] which the stability checker returns as $|b| < 1 - |a|$ (note that the first condition $|a| < 1$ is actually redundant). By evaluating `Assuming[All]`, the MDSPPs will list the assumptions it has made about free parameters In the output of `Assuming[All]` above, the first two inequalities are the original inequality conditions generated from the ROC. The third inequality arose during the simplification of the second inequality.

Figure 5.7: Stability Analysis of a Non-Separable Two-Dimensional Signal

multidimensional unit circle: $|z_1| = 1$, $|z_2| = 1$, etc. In N dimensions with z-transform variables $\{z_i\}_{i=1}^{N}$, the comparison test for stability [53] generalizes to

$$\max_{\substack{i=1\ldots N,\, i\neq j \\ |z_i|=1}} R_- < 1 < \min_{\substack{i=1\ldots N, i\neq j \\ |z_i|=1}} R_+ \quad \text{where} \quad R_- < |z_j| < R_+ \quad \text{for} \quad j = 1 \ldots N$$

Similarly for the Laplace transform of continuous-time signals, the stability test generalizes to

$$\max_{\substack{i=1\ldots N,\, i\neq j \\ \Re e(s_i)=0}} R_- < 0 < \min_{\substack{i=1\ldots N, i\neq j \\ \Re e(s_i)=0}} R_+ \quad \text{where} \quad R_- < \Re e(s_j) < R_+ \quad \text{for} \quad j = 1 \ldots N$$

The stability checking routine replaces any $\Re e(s_i)$ terms in the Laplace transform ROC with zero.

5.5.2 Graphical Analysis of Stability

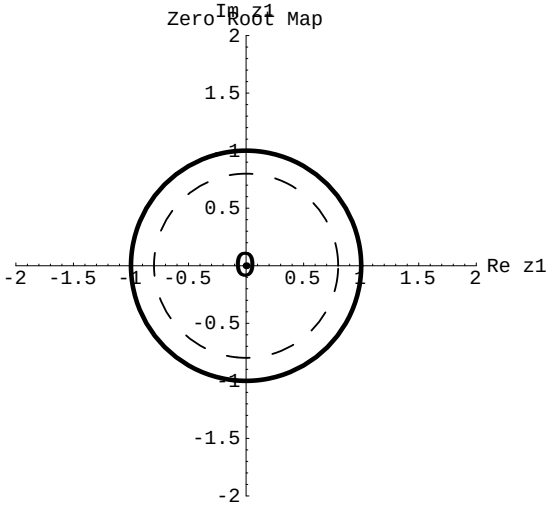
The results of transforms are sometimes best understood in terms of graphical representations. A wide array of graphical capabilities have been implemented in the MD-SPPs for 1-D and 2-D signals in both the discrete-time and continuous-time domains: continuous and discrete plots, magnitude and phase plots, and pole-zero diagrams. For non-separable 2-D systems, a pole-zero diagram can be displayed as the locus of poles and zeros of one transform variable parameterized by the other [53]. By inspection of the loci of poles and zeros, one can determine the stability of the signal by using the same rules as in one dimension: if the poles lie outside of the unit circle, then the causal implementation of the system would be unstable. Underlying the pole-zero root loci diagrams is a root locus plotting command for one free variable.

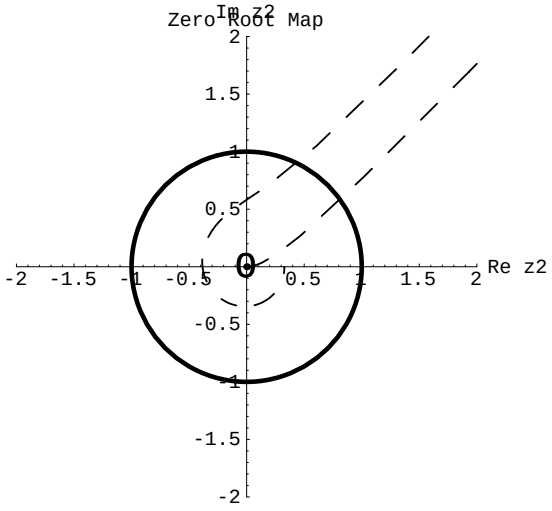
Figure 5.8 analyzes the location of the poles and zeros for a signal. Zeros are shown as **O**'s, poles are shown as **X**'s, the unit circle is shown as a solid curve, and the lower bound of the ROC R_- is graphed with dashed lines. The upper bound of the ROC R_+ is not shown because it is infinite for both transform variables. The left column of Figure 5.8 shows the zero and pole diagrams of the second transform variable `z2` projected onto `z1` by the mapping $z_2 = \exp(j\omega_2)$. The right column

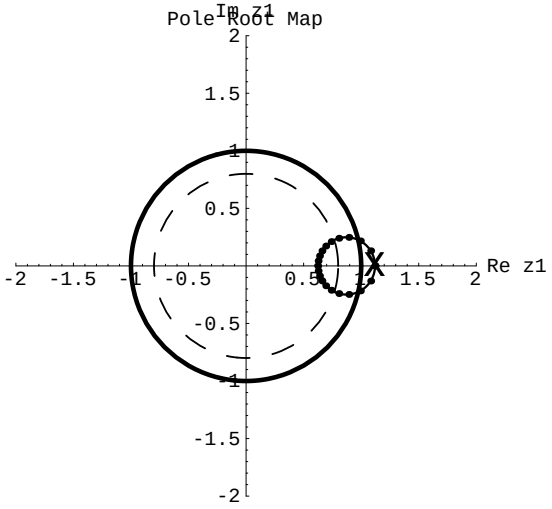
```
PoleZeroPlot[
     ZTransform[ (n1+n2)! (4/5)^n1 (2/7)^n2 Step[n1,n2] / (n1! n2!),
                 {n1, n2}, {z1, z2} ] ]
```

```
                                  I w
Numerator polynomial in z1: 35 E     z1
Denominator polynomial in z1:

      I w                 I w
  -28 E     - 10 z1 + 35 E     z1
```

```
                                  I w
Numerator polynomial in z2: 35 E     z2
Denominator polynomial in z2:

      I w                 I w
  -10 E     - 28 z2 + 35 E     z2
```







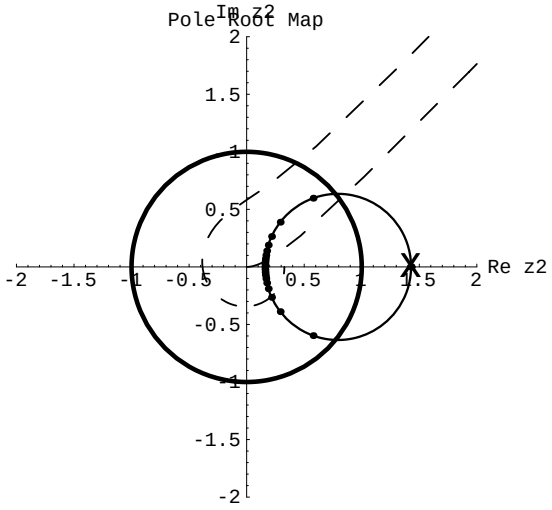


Figure 5.8: Two-Dimensional Pole-Zero Diagrams for an Unstable Signal
The lower limit of the region of convergence (ROC) is plotted as dashed lines. The ROC extends to infinity in both z variables. In parts of the root loci, the poles and zeros actually move outside of the ROC because the system is unstable.

of Figure 5.8 shows the zero and pole diagrams of the first transform variable `z1` projected onto `z2` by the mapping $z_1 = \exp(j\omega_1)$. Since the poles fall outside of the unit circle, the signal is unstable.

5.6 Analysis of Multidimensional Multirate Systems Using Transforms

This section discusses the use of the MDSPPs to analyze multidimensional multirate systems. First, Section 5.6.1 applies general two-dimensional signal analysis to a downsampled signal. Second, Section 5.6.2 shows how the MDSPPs can help in the visualization of aliasing in two dimensions. Last, Section 5.6.3 performs a purely symbolic analysis of resampling.

5.6.1 Automated Two-Dimensional Signal Analysis

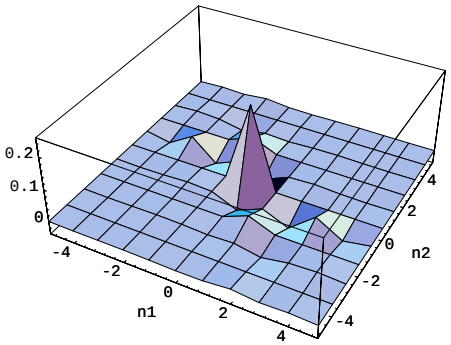
We now elaborate on the two-dimensional signal analysis abilities introduced in previous sections of this chapter. *Mathematica* can already plot two-dimensional continuous-time functions as three-dimensional mesh plots via its `Plot3D` command. `Plot3D` can also be used to generate the mesh plots for two-dimensional sequences. For two-dimensional sequences, however, a better alternative is to use `DensityPlot` to render an aerial view of the domain and range (see Figure 4.1). In two dimensions, pole-zero diagrams for z-transforms become root maps [53], as mentioned in Section 5.5.2. Likewise for two-dimensional Laplace transforms, the pole-zero diagram is a root map— s_1 is plotted with $s_2 = j\omega_1$ for different values of ω_1 and visa-versa for s_2. Frequency responses are plotted using a parametric plotting routine. Figure 5.9 performs analysis on a discrete-time signal which suffers from aliasing distortion.

```
lowpass2Dtile =
   CPulse[Pi, w1 + Pi/2] CPulse[Pi, w2 + Pi/2];
lowpass2D =
   InvDTFTransform[ lowpass2Dtile, {w1, w2} ]
```

$$\frac{\mathrm{Sinc}[\frac{n1\ Pi}{2}]\ \mathrm{Sinc}[\frac{n2\ Pi}{2}]}{4}$$

```
downMatrix = {{2, 1}, {1, 3}};
DSPAnalyze[
   Downsample[downMatrix, {n1,n2}][lowpass2D],
   {n1, n2}, {-5, -5}, {5, 5} ]
```

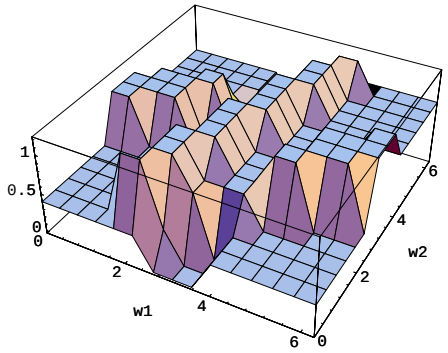
Discrete-Time Domain Analysis



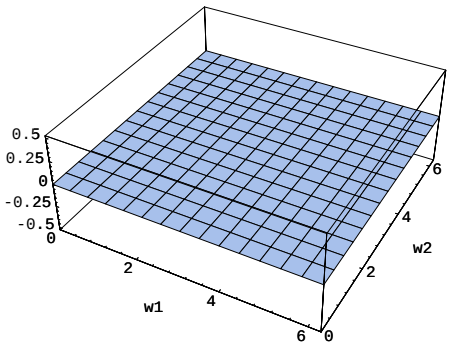

```
Transform::incomplete:
   The rule base could not compute the forward
                          n1
    z-transform of Sinc[--] with respect to n1.
                          2

ZTransform::notvalid:
   The forward z-transform could not be found.

                              n1 Pi       n2 Pi
                           Sinc[-----] Sinc[-----]
                                2           2
Downsample                [-----------------------]
          2   1  |  n1              4

          1   3  |  n2

has the following frequency response:

                                        Pi   3 w1   w2
Periodic                      [(CPulse  [-- + ---- - --]
        2 Pi   0     |  w1          Pi 2     5     5

        0      2 Pi  |  w2

             Pi   w1   2 w2
    CPulse  [-- - -- + ----] +
          Pi 2    5     5

            Pi   3 (-4 Pi + w1)   -6 Pi + w2
   CPulse  [-- + -------------- - ----------]
         Pi 2          5              5

             Pi   -4 Pi + w1   2 (-6 Pi + w2)
    CPulse  [-- - ---------- + --------------] +
          Pi 2         5              5

            Pi   3 (-4 Pi + w1)   -4 Pi + w2
   CPulse  [-- + -------------- - ----------]
         Pi 2          5              5

             Pi   -4 Pi + w1   2 (-4 Pi + w2)
    CPulse  [-- - ---------- + --------------] +
          Pi 2         5              5
```

```
            Pi   3 (-2 Pi + w1)   -4 Pi + w2
CPulse  [-- + -------------- - ----------]
      Pi 2          5              5

            Pi   -2 Pi + w1   2 (-4 Pi + w2)
 CPulse  [-- - ---------- + --------------] +
       Pi 2         5              5

            Pi   3 (-2 Pi + w1)   -2 Pi + w2
CPulse  [-- + -------------- - ----------]
      Pi 2          5              5

            Pi   -2 Pi + w1   2 (-2 Pi + w2)
 CPulse  [-- - ---------- + --------------]) / 5]
       Pi 2         5              5
```

Magnitude Response



Phase Response (degrees)



-Incomplete z-Transform-

The first command defines a 2-D time-domain signal that is an ideal quarter-band lowpass filter. The second command invokes the digital signal analyzer. The analyzer first plots the time response, then reports that it cannot find the z-transform because of the two-sided sinc functions, and finally computes and plots the DTFT. Downsampling by 5 quintuples the area of support in the the frequency domain which causes aliasing. Aliasing appears in the magnitude plot for amplitudes above 0.2.

Figure 5.9: Signal Analysis of Aliasing in Two Dimensions

5.6.2 Visualization of Downsampling in Two Dimensions

We will now use the MDSPPs to visualize and analyze the effects of downsampling by matrix M on a baseband signal $X(\omega)$. The baseband signal $X(\omega)$ is periodic with period of 2π in each discrete-time frequency variable. In terms of the discrete-time frequency variables ω, downsampling by M periodically replicates the baseband $|\det M|$ times in each $2\pi \times \cdots \times 2\pi$ frequency tile. The centers of the new frequency bands occur at the original center of the baseband shifted by the aliasing vectors associated with M (see Section 3.1). The aliasing vectors, computed according to equation (3.5), are: $\omega_i = 2\pi \left(M^T\right)^{-1} \mathbf{k}_i$, where $i = 0, \ldots, |\det M| - 1$ and $\mathbf{k}_i$ is the ith coset vector.

In the MDSPPs, the `DownsamplingAnalysis` routine determines if the downsampled signal experiences aliasing and if it covers the frequency domain. A downsampled signal can suffer from two kinds of aliasing: intra-band aliasing and inter-band aliasing. In intra-band aliasing, a frequency band overlaps with itself, whereas in inter-band aliasing, a frequency band overlaps with another frequency band. Clearly, if a downsampled signal does not experience aliasing and covers the frequency domain, then the signal has been resampled at its Nyquist rate.

`DownsamplingAnalysis` represents the downsampling operation by its downsampling matrix and the baseband signals as a list of polygons. For example, Figure 5.10 shows the effect of downsampling by a quincunx downsampling matrix on a diamond-shaped passband. A quincunx resampling matrix has a determinant of 2 or -2 and non-zero entries. The lattice (sampling grid) generated by quincunx matrices correspond to diagonal interleaving, which is commonly found in video signals. The vertices of the diamond-shaped passband are obtained by mapping a separable low-pass filter by a quincunx matrix. In Figure 5.10, the quincunx downsampler resamples the baseband signal at its Nyquist rate.

In Figure 5.11, we keep the same baseband but use a different downsampling matrix. The downsampling matrix has a determinant of three. Since the baseband

```
DownsamplingAliasing[
  {{1, 1}, {1, -1}},
  Polygon[ {{-Pi,0}, {0,Pi}, {Pi,0}, {0,-Pi}} ],
  Dialogue -> All ]
```

Analyzing aliasing for the downsampling matrix

$$\begin{matrix} 1 & 1 \\ 1 & -1 \end{matrix}$$ for a baseband whose domain is

described by the polygon with vertices

{{-Pi, 0}, {0, Pi}, {Pi, 0}, {0, -Pi}}

The downsampling will yield the baseband plus 1 shifted/skewed copy of the baseband.

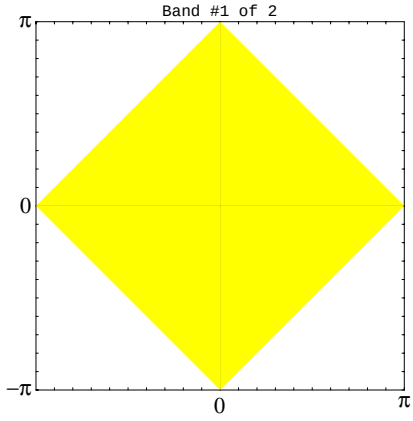
The shifting vectors are {{0, 0}, {Pi, Pi}}

The area of the baseband is

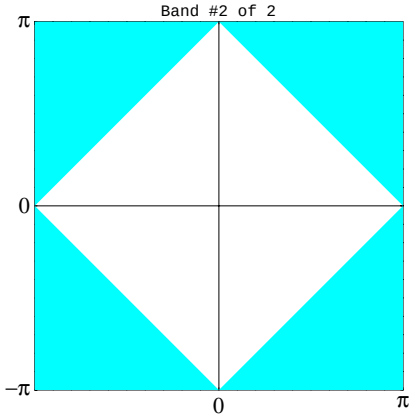
$2\,Pi^2$ whose numerical value is 19.7392.

Initial analysis: the downsampler resamples the baseband signal at its Nyquist rate (assuming that no aliasing is present).

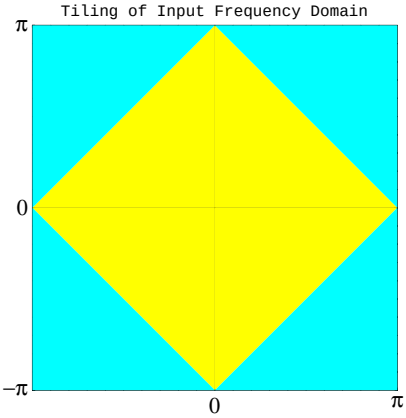
Band #1 of 2 is free of intra-band aliasing.



Band #2 of 2 is free of intra-band aliasing.



Tiling of input frequency domain for the 2 bands:



The downsampler does not introduce aliasing.

It resamples the baseband at the Nyquist rate.

Figure 5.10: Quincunx Downsampling Without Aliasing

occupies 50% of the frequency domain, generating three copies of the baseband will produce inter-band aliasing. `DownsamplingAliasing` reports that no intra-band aliasing occurs and that inter-band aliasing occurs in 50% of the frequency domain.

The `DownsamplingAliasing` routine works as follows. For each new band, the routine computes the location of the vertices of the band, decomposes the band into a union of triangles, and determines how each triangle maps into the fundamental frequency tile $\omega_1 \in (-\pi, \pi) \bigcup \omega_2 \in (-\pi, \pi)$ by calling the `TriangleModWithSquare` routine. The `TriangleModWithSquare` routine is recursive. For each triangle,

- if each of the triangle vertices is either inside or on the border of the fundamental tile, then return the triangle;
- if one or more of the corners of the fundamental tile is inside the triangle, then split the triangle into several triangles by using the corner(s) of the fundamental tile inside of it as the new vertices and pass each new triangle to `TriangleModWithSquare`;
- if the triangle is completely outside of the fundamental tile, then shift the triangle by the right number of periods to put at least one vertex in the fundamental tile and pass it to `TriangleModWithSquare`; and
- if none of the above cases apply, then the triangle intersects one or more of the edges of the fundamental tile, so we split the triangle at each edge intersection into smaller triangles:
  - return the triangles inside the fundamental tile, and
  - pass triangles outside of the fundamental tile to `TriangleModWithSquare`.

Once the new band has been generated, the `DownsamplingAliasing` routine checks for intra-band aliasing by checking for overlapping regions in the band (via the `OverlappingRegions` routine). The new band is then plotted with intra-band aliasing region (if it exists) shaded black. After all the bands have been generated, the

```
DownsamplingAliasing[
  {{1, 1}, {2, -1}},
  Polygon[ {{-Pi, 0}, {0, Pi}, {Pi,0}, {0,-Pi}} ],
  Dialogue -> True ]
```

Analyzing the aliasing for the downsampling matrix

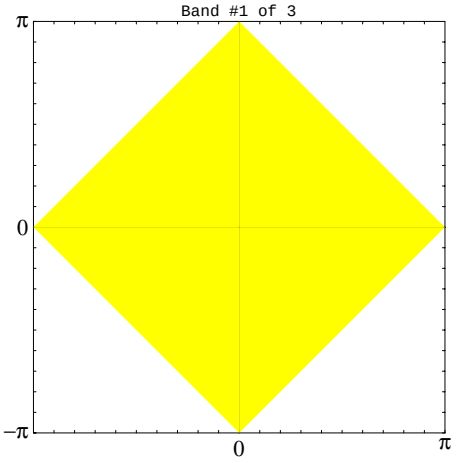
$$\begin{matrix} 1 & 1 \\ 2 & -1 \end{matrix}$$

for a baseband whose domain

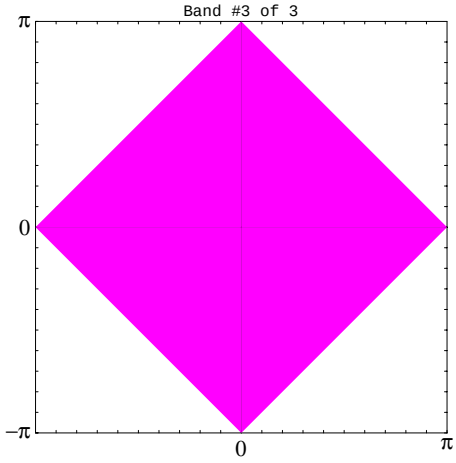
is described by the polygon with vertices

{{-Pi, 0}, {0, Pi}, {Pi, 0}, {0, -Pi}}

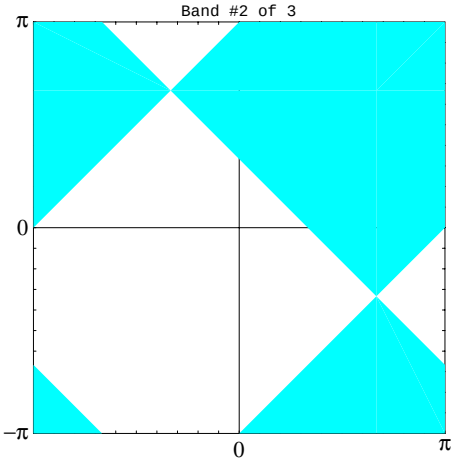
Band #1 of 3 is free of intra-band aliasing.



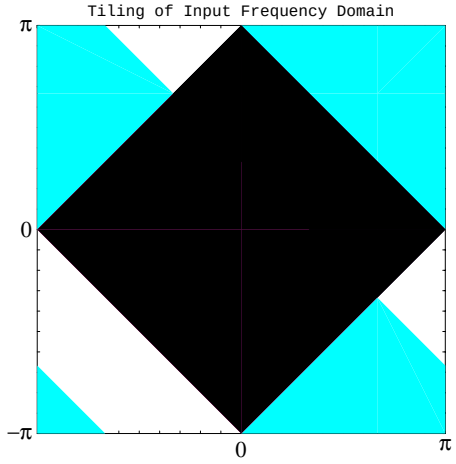
Band #3 of 3 is free of intra-band aliasing.



Band #2 of 3 is free of intra-band aliasing.



Tiling of the input frequency domain for the 3 bands with inter-band aliasing shown in black:



The downsampler introduces inter-band aliasing.
Aliasing occurs in 50% of the frequency domain.

Figure 5.11: Quincunx Downsampling With Aliasing

`DownsamplingAliasing` routine checks for inter-band aliasing by checking for overlapping regions between bands. The routine then computes the total area of the domain in which aliasing occurs.

5.6.3 Automatic Derivation of Transform Properties

When all the free parameters have been assigned values, graphical-based signal analysis provides much insight. Sometimes, symbolic analysis can help in the choice of values for free parameters. For example, Section 5.4 showed that the transform routines can derive the input-output relationship of a system in a transform domain if "abstract" transform pairs such as $x[n] \longleftrightarrow X(z)$ are provided. In this section, we examine the use of abstract transform pairs to derive properties of the transforms of multidimensional multirate signals.

When upsampling a signal in two dimensions say by some upsampling matrix $\begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix}$, the resulting transform is

$$X\left(z_1^{u_{11}} z_2^{u_{21}}, z_1^{u_{12}} z_2^{u_{22}}\right)$$

given that the z-transform of the original signal is $X(z_1, z_2)$. The MDSPPs can generate this relationship by specifying the abstract transform pair $x[n_1, n_2] \longleftrightarrow X(z_1, z_2)$:

```
In[2]:= upMatrix = {{u11, u12}, {u21, u22}};

In[3]:= ZTransform[ Upsample[ upMatrix, {n1, n2} ] [ x[n1,n2] ],
                    {n1,n2}, {z1, z2},
                    TransformLookup -> { x[n1,n2] :> X[z1,z2] } ]

                       u11   u21    u12   u22
Out[3]= ZTransData[X[z1    z2   , z1    z2   ], Rminus[{0, 0}],

>    Rplus[{Infinity, Infinity}], ZVariables[{z1, z2}]]
```

The MDSPPs will also track the region of convergence of the upsampled signal if we give the ROC of the input signal:

```
In[4]:=
   ZTransform[
      Upsample[ upMatrix, {n1, n2} ] [ x[n1, n2] ],
      {n1, n2}, {z1, z2},
      TransformLookup ->
         { x[n1,n2] :> { X[z1,z2], {rm1,rm2}, {rp1,rp2} } } ]

                         u11   u21    u12   u22
Out[4]= ZTransData[X[z1    z2   , z1    z2   ],

                 1/u11    1/u21     1/u12    1/u22
>    Rminus[{rm1      rm2     , rm1      rm2     }],

                1/u11    1/u21     1/u12    1/u22
>    Rplus[{rp1      rp2     , rp1      rp2     }],

>    ZVariables[{z1, z2}]]
```

The resulting multidimensional ROC is not separable. For each zero element u_{ij} in the upsampling matrix, the corresponding $x^{u_{ij}}$ and $x^{1/u_{ij}}$ terms equal 1 for any value of x. When the input signal is stable, the ROC of the z-transform of the upsampled signal will be a compressed version of the ROC of the z-transform of the input signal because of the nth root terms. The MDSPPs can track the ROC in higher dimensions.

5.7 Summary

This chapter first identifies important properties of multidimensional signals based on the properties of one-dimensional signals used in SPLICE and ADE and then discusses the high-level capabilities of the signal processing packages. These high-level routines, which are summarized in Table 5.2, rely on the representation of signals and systems as functions and operators, respectively, as discussed in Chapter 4. The high-level routines cover a wide range of graphical and symbolic analyses for both

New Function	Dimension of Signal	Description
`ASPAnalyze`	1-D and 2-D	general analog signal analyzer
`CTFTransform`	1-D and m-D	forward continuous-time Fourier transform
`DFTransform`	1-D and m-D	forward discrete Fourier transform
`DTFTransform`	1-D and m-D	forward discrete-time Fourier transform
`DSPAnalyze`	1-D and 2-D	general digital signal analyzer
`InvCTFTransform`	1-D and m-D	inverse continuous-time Fourier transform
`InvDFTransform`	1-D and m-D	inverse discrete Fourier transform
`InvDTFTransform`	1-D and m-D	inverse discrete-time Fourier transform
`InvLaPlace`	1-D and m-D	inverse Laplace transform
`InvZTransform`	1-D and m-D	inverse z-transform
`LaPlace`	1-D and m-D	forward Laplace transform
`LSolve`	1-D	linear constant coefficient differential equations solver
`PiecewiseConvolution`	1-D	discrete and continuous convolution
`PiecewisePlot`	1-D	piecewise function plotter
`PoleZeroPlot`	1-D and 2-D	display of pole-zero diagrams
`RootLocus`	1-D	root locus of one free parameter
`SequencePlot`	1-D and 2-D	digital signal plotter
`SignalPlot`	1-D and 2-D	analog signal plotter
`Stable`	1-D and m-D	stability checker
`ZSolve`	1-D	linear constant coefficient difference equations solver
`ZTransform`	1-D and m-D	forward z-transform

Table 5.2: High-Level Abilities of the Signal Processing Packages

discrete-time and continuous-time signals and systems. The high-level capabilities work on non-separable multidimensional signals and systems, except for the convolution, root locus, and differential/difference equation solving routines which only work for separable ones. The extended graphics routines help engineers to visualize one-dimensional and two-dimensional signals and systems. The transform, convolution, and differential/difference equation (DE) solvers can provide closed-form symbolic answers to many problems that would otherwise be difficult and prone to error when worked by hand. In addition, these routines can show how they compute the answer in a step-by-step manner for verification. This dialogue is in natural language for the transform and DE solvers, but it takes the form of animation for convolution. We have combined graphical and symbolic analyses for one-dimensional and two-dimensional signals into two general signal analyzers, one for continuous-time and one for discrete-time signals.

CHAPTER 6

Rearranging Multidimensional Systems

This chapter discusses the rearrangement of algorithms with the goal of finding better or even optimal implementations. By rearranging components of an algorithm, E-SPLICE and ADE have derived efficient implementations for three specific one-dimensional multirate systems [31]. E-SPLICE derived the fact that the cascade of an upsampler and downsampler commutes if their resampling factors are relatively prime [9]. E-SPLICE and ADE have generated polyphase forms for systems to change the sampling rate by rational factors [9, 11]. ADE has found an efficient multiband structure for the discrete-time implementation of optimal detectors (frequency-chirp matched filters) of FSK-coded sonar signal beams based on the pruned FFT [10, 12].

E-SPLICE and ADE know how to rearrange cascade and parallel combinations of linear one-dimensional (multirate) operators. Many of the rules are based on operator properties such as linearity. Section 6.1 extends these properties to multidimensional multirate operators and identifies new properties that became important in multiple dimensions. E-SPLICE and ADE do supplement their property-based rules to describe how to rewrite combinations of operators, especially between resamplers and other operators. We have identified the multidimensional versions of these rules [39, 52, 53, 71, 72] and encoded them in the multidimensional signal processing packages (MDSPPs), as discussed in Chapter 3. Section 6.2 describes different strategies to apply a set of rearrangement rules.

E-SPLICE and ADE utilize cost functions to rank the efficiency of equivalent implementations of an algorithm. E-SPLICE measures the number of additions and the number of multiplications, whereas ADE also takes the amount of memory needed into account. Given a cost function, rearrangement rules can be applied until a

better or even optimal implementation is discovered. ADE uses heuristics to reduce the number of equivalent implementations (called the equivalence space) that it generates. The MDSPPs use heuristics as well, as discussed in Section 6.3.

6.1 Extending System Properties in E-SPLICE and ADE

In order to prevent a combinatoric explosion in the number of rearrangement rules, E-SPLICE and ADE base many of their rearrangement rules on system properties such as those given earlier in Table 2.2 (c.f. Appendix D of [8]). These system properties extend naturally to higher dimensions. The MDSPPs add the properties `DISCRETE` and `CONTINUOUS` to distinguish the domain of the input signals. This was not an issue in E-SPLICE and ADE because they only supported discrete-time signals. We have also introduced the properties `LINEARPHASE` and `DELAY` because they are important parameters in image processing and communications, respectively. Another new property, `SEPARABLE`, indicates if a multidimensional operation can be decomposed into separable one-dimensional operations. Table 6.1 lists the properties that the MDSPPs implement.

To complement their property-based rules, E-SPLICE and ADE introduce rules to capture special relationships between operators. The rules containing operators that are separable in multiple dimensions can be updated easily. Most of the rules, however, do not fall into this category because they contain downsampling, upsampling, or filtering operations which are not always separable in multiple dimensions. Chapter 3 discusses in detail the rules for rewriting cascades of multidimensional multirate operations, which have been encoded in the MDSPPs.

System Property	Meaning
ASSOCIATIVE†	can change grouping of inputs
ADDITIVE†	distributes over addition
COMMUTATIVE†	can change order of inputs
CONTINUOUS	inputs are continuous signals
DELAY	amount of delay before output is meaningful
DISCRETE	inputs are discrete signals
HOMOGENEOUS†	scaled input gives scaled output
LINEAR†	additive and homogeneous
LINEARPHASE	true if the frequency phase response is a linear function of the frequency variable
MEMORYLESS†	output does not depend on previous inputs or outputs; if a single-input system, then SHIFTINVARIANT
SEPARABLE	true if separable in all dimensions, false if completely non-separable, or a list of variables in which the operator is separable
SHIFTINVARIANT†	shifted input gives shifted output

† property also included in E-SPLICE [8]

Table 6.1: System Properties in the MDSPPs

6.2 Algorithm Rearrangement

In rearranging operations in an algorithm, both simplification and rearrangement rules are applied to the algorithm. Simplification rules produce more efficient algorithms through the removal of one or more components (e.g. a shift by l undoes a shift by $-l$). Rearrangement rules produce different but not necessarily better implementations. For linear systems, some rearrangement rules can be based on system properties (e.g. the order of two linear shift-invariant operators can be switched), but most rules are expressed in terms of specific operators (e.g. the cascade of two shift operators can be rewritten as one shift operator whose shift is the sum of the two original shift factors).

6.2.1 General Procedure

The procedure to rewrite algorithms is iterative. At each iteration, the first step is to determine which rearrangement rules apply, beginning at the output(s) of the algorithm. The search for applicable rules may continue until the input(s) to the algorithm is reached. Then, the algorithm is rewritten by applying one or more of the rules. Finally, at each iteration, each equivalent form of the algorithm is simplified. The end result is a tree of equivalent forms.

In determining which rules may apply to an algorithm, rules that would produce an expression that has already been generated would not be considered. If maintaining parallelism in a part of the algorithm is important, then rules would have to apply to every branch in a parallel structure (called a regularity constraint in ADE). Once a collection of candidate rules has been determined, the procedure could apply all of them to the algorithm which would likely create a tree that would grow exponentially in size. The other extreme is to apply only one rule at each iteration (e.g. choose one at random). These two opposing strategies correspond to a breadth-first search and a depth-first search, respectively, through the equivalence space (the tree of equivalent

implementations). Clearly, the second approach would be sub-optimal, but the first approach may not find the optimal solution in a reasonable amount of time.

6.2.2 Rearranging A Multidimensional Rational Rate Changer

A simple example of algorithm rearrangement is a structure to change the two-dimensional sampling rate first by upsampling the input signal $x[n_1, n_2]$ by $\begin{bmatrix} 6 & 8 \\ 3 & 3 \end{bmatrix}$ and then by downsampling the result by $\begin{bmatrix} 9 & 12 \\ 7 & 7 \end{bmatrix}$. In the MDSPPs, the structure would be written as a nested algebraic expression:

```
Downsample[ {{9, 12}, {7, 7}}, {n1,n2} ] [
     Upsample[ {{6, 8}, {3, 3}}, {n1,n2} ][ x[n1,n2] ] ]
```

The rearrangement rule shown in Figure 3.12(a) will seek to decompose up/downsampling cascades by first factoring the up/downsampling matrices into their Smith forms and then by removing redundant resampling operations. In this case,

$$\begin{bmatrix} 6 & 8 \\ 3 & 3 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 3 & 4 \\ 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 9 & 12 \\ 7 & 7 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 0 & 7 \end{bmatrix} \begin{bmatrix} 3 & 4 \\ 1 & 1 \end{bmatrix}$$

so the common right unimodular matrix can be removed, thereby making the resampling separable:

```
Downsample[ {{3, 0}, {0, 7}}, {n1,n2} ] [
     Upsample[ {{2, 0}, {0, 3}}, {n1,n2} ][ x[n1,n2] ]
```

Another rearrangement rule decomposes separable multidimensional resampling operations into one-dimensional resampling operations:

```
Downsample[3, n1][ Downsample[7, n2][
     Upsample[2, n1][ Upsample[3, n2][ x[n1,n2] ] ] ] ]
```

Applying the rearrangement rule that commutes downsamplers in cascade and another that commutes upsamplers in cascade, we ultimately get

```
Upsample[2, n1][ Downsample[3, n1][
     Upsample[3, n2][ Downsample[7, n2][ x[n1, n2] ] ] ] ]
```

The overall rate change is the product of the original downsampling matrix and the inverse of the upsampling matrix:

$$\begin{bmatrix} 9 & 12 \\ 7 & 7 \end{bmatrix} \begin{bmatrix} 6 & 8 \\ 3 & 3 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{3}{2} & 0 \\ 0 & \frac{7}{3} \end{bmatrix}$$

6.3 Finding Optimal Algorithms

Adding the ability to measure the performance of equivalent forms of algorithms can help guide the rearrangement process to search for better implementations. At each iteration in the depth-first search strategy for applying rearrangement rules, the lone rule could be chosen as the one among the candidates that produces the best new implementation. When no more rules apply, the best implementation would be chosen from all of the equivalent forms generated. For the breadth-first search strategy, candidate rules could be pruned by only selecting those that will produce an algorithm with a lower cost or increase the cost over the current (or original) algorithm by no more than some factor.

Both E-SPLICE and ADE implement cost functions to rank equivalent algorithms [77]. E-SPLICE counts only the number of additions and multiplications times the sampling rate. ADE also counts the number of memory elements (which is independent of the sampling rate). The MDSPPs measure all three costs. In multiple dimensions, however, the sampling rate is no longer a real number but instead a real-valued matrix. The MDSPPs measure the number of additions and multiplications times the average output sampling rate.

Unlike E-SPLICE and ADE, the MDSPPs can associate a range of values with a free parameter. The implementation cost of linear operators for each free parameter either monotonically increases or monotonically decreases for each free parameter.

When one free parameter is allowed to take values over an interval, the associated implementation cost for an operator depending on that free parameter also takes values over an interval. Even when an operator depends on two or more such free parameters, the cost function still takes values over an interval. As a consequence, the implementation cost of two algorithms can be compared in terms of intervals.

6.4 Summary

The primary goal behind rearranging the form of an algorithm is to find a better implementation of algorithm. To find better implementations, a computer program needs a set of simplification and rearrangement rules as well as a function to measure the implementation cost of an algorithm. When applied, simplification rules will produce a better implementation because they remove inverse operations that appear in cascade. Rearrangement rules, on the other hand, simply replace a part of an algorithm by an equivalent form that may or may not be more efficient.

E-SPLICE and ADE express some rearrangement rules for one-dimensional systems in terms of system properties, but the majority are based on specific operators. Since system properties (such as linearity) extend to higher dimensions, we have recast the rearrangement rules based on system properties for multidimensional systems. We also add new properties and base new rules on them. In Chapter 3, we derive the special case rules in multiple dimensions. We also extend the simplification rules present in E-SPLICE and ADE to multiple dimensions.

In computing computational cost for one-dimensional multirate algorithms, the number of additions and multiplications is scaled by the sampling rate. For multiple dimensions, the computational cost depends instead on the sampling matrix instead of the sampling rate. The other costs (such as the amount of memory elements) are independent of the sampling rate and extend naturally to higher dimensions.

CHAPTER 7

Generating Equivalent Code for Algorithms

As the previous two chapters show, symbolic reasoning can assist the engineer in designing algorithms with a relatively small number of free parameters and interconnections (see Figure 5.6, for example). Because we implement reasoning on mathematical formulas, we do not have a convenient way to represent the many connections in complex systems. The layout of complex systems is better captured graphically by tools that can organize block diagrams hierarchically.

Generating equivalent code for developed algorithms is essential to transport the algorithm to another environment for layout, typesetting, further simulation, and so forth. Specifically, the MDSPPs translate signal processing expressions into TEX document processing commands [78] and Ptolemy block diagram representations [13, 14, 15, 16]. We have chosen TEX because it is a typesetting tool commonly used by engineers to document designs, write reports, etc. We have chosen the Ptolemy environment because it can represent complex systems and simulate algorithms as well as generate C programs, DSP assembly language code, and VHDL descriptions. The ability to generate Ptolemy code places the MDSPPs in the rapid prototyping process shown earlier in Table 1.1.

7.1 Generating TEX Code

TEX, a typesetting language developed by Donald Knuth, formats text and equations in a standard way for publication. When formatting equations, TEX displays mathe-

matical formulas according to the conventions defined by the American Mathematical Society. When typesetting text, TEX follows standard rules followed by printers. We have chosen TEX as a target environment because it is commonly used by engineers to document projects, write technical reports, etc.

An inherent ability of *Mathematica* is to translate a single algebraic expression into a TEX equation. These equations can then be spliced into existing documents. Ideally, this automatic translation eliminates the possibility of errors that can occur while retyping a formula.

For example, `TeXForm[t^2]` gives `{t^2}` which is interpreted as t^2 once the expression is put into a TEX math environment (i.e. surrounded by dollar signs). A more complicated example is the input-output relationship of a two-channel non-uniform filter bank shown in Figure 5.6(c). We first had *Mathematica* factor the expression in a meaningful way and then called `TeXForm` to generate the equation.

In the extended *Mathematica* environment, the multidimensional signal processing packages contain the TEX definitions for signals and systems they introduce. For example,

```
TeXForm[ Downsample[2,n][x[n]] ]
```

generates one line of TEX code `\downarrow_{2,n}(x[n])` which, after TEX processing, gives $\downarrow_{2,n}(x[n])$. For a more complicated example, we will use the time domain description of the two-channel non-uniform filter bank shown in Figure 5.6(b):

```
TeXForm[ upperchannel + lowerchannel ]
```

gives

$$g1(n) \star_n \uparrow_{3,n} (\downarrow_{3,n} (h1(n) \star_n x(n))) + \downarrow_{2,n} (g0(n) \star_n \uparrow_{3,n} (\downarrow_{3,n} (h0(n) \star_n \uparrow_{2,n} (x(n)))))$$

Note that $\star_n$ means convolution in the variable n. The impulse response of the filters are not subscripted here as we did not encode a rule to print `g0` as g_0.

7.2 Generating Complete Ptolemy Simulations

We now consider issues involved in converting a set of mathematical formulas into a complete working program. In our case, we are ultimately concerned with converting a set of mathematical formulas into its Ptolemy block diagram specification with the option of generating additional code to direct the running of a complete simulation, as is discussed in Section 7.2.2. First, Section 7.2.1 examines the general problem of converting formulas to a complete high-level program.

7.2.1 Program Synthesis

Like its ability to generate TEX code, *Mathematica* can convert a formula into its equivalent C or Fortran code via the `CForm` and `FortranForm` commands, respectively. With a high-level programming language comes the additional problem of linking routines to libraries. Code generated by *Mathematica* will have the correct syntax, but it may not contain valid function calls. For example, `CForm[t^2]` produces Power(t,2). In order for such a statement to compile properly, one must either define a C macro to preprocess Power into the proper function say pow or provide a library function called Power. Alternately, one can program *Mathematica* to generate the proper function call. In our example, we want `CForm` to translate the power function in *Mathematica* to the pow function in the C math library:

```
Unprotect[Power, pow];
Clear[pow];
Format[Power, CForm] := pow;
Protect[Power, pow];
```

Now, `CForm[t^2]` would return pow(t,2).

Even once we provide all of these missing "hooks" to the target language, we can only generate lines of code that will compile properly. We still do not have the ability to generate subroutines, and therefore, we cannot create complete programs from

scratch. SINAPSE, developed at the Schlumberger Laboratory for Computer Science, is an example of a system that uses *Mathematica* to generate complete programs [79, 80, 81]. SINAPSE translates a set of partial differential (wave) equations with boundary conditions into finite difference approximations. From the finite difference approximations, SINAPSE generates optimized Fortran-77, Connection Machine^{TM} Fortran, or C source code. Complete source code generation takes on the order of 10 minutes on a Sun SparcStation 2. SINAPSE directs the user in choosing the wave phenomena, the boundary conditions, the finite difference technique, the target language, and the target machine (either a sequential computer or the parallel Connection Machine).

7.2.2 Converting Algebraic Formulas to Working Ptolemy Simulations

As previously mentioned, we would like to translate signal processing algorithms expressed as mathematical formulas to the Ptolemy environment so that the MDSPPs can fit into a rapid prototyping process. Ptolemy has both a graphical user interface and a command line interpreter. The command line interpreter is the actual target for code generation.

Ptolemy represents systems using block diagrams. Unlike a high-level language, Ptolemy does not support nested function calls. Therefore, we must unravel nested calls in an algebraic formula to decompose it into a sequence of simple block operations. A simple block operation consists of a set of inputs, one function call, and a set of outputs.

In *Mathematica*, the `PtolemyProgram` command converts an algebraic formula to a complete Ptolemy simulation; its algorithm is given in Figure 7.1. The order and syntax of the information given to the `PtolemyProgram` routine mimics the calling sequence of the various plotting commands in *Mathematica*:

$$routine\ [\ expression,\ \{v,\ v_{min},\ v_{max}\},\ options\ ]$$

The pre-processing converts impulse responses of filters that appear in convolution operators into filtering operations and maintains the names of the original constants so that they will appear as the same name in the Ptolemy code. The Ptolemy code can then be edited at a future time to insert new values for the constants.

Since the generated Ptolemy program will ultimately be interpreted and run, we must specify a header for the program that will define the signals and systems commonly used in the MDSPPs but missing in Ptolemy's library. We provide a default header simply called "header.pt" which will either be loaded by the Ptolemy program as its first step or be copied verbatim to the beginning of the Ptolemy program (the default). In the first case, the global parameter `$PtolemyProlog` must be set to

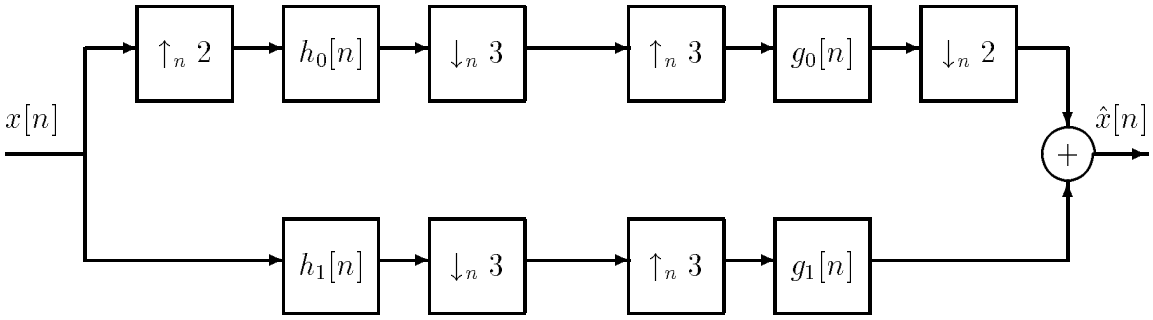
```
StringJoin[ "(load \"",
            FindFile[ "SignalProcessing`ObjectOriented`header.pt",
                      $Path ],
            "\")" ]
```

The default behavior allows the output of the `PtolemyProgram` routine to be piped to a computer (possibly on a separate network) that runs Ptolemy. Ptolemy runs simulations by incrementing a floating-point counter over a range of values. As a part of the code generation, we define the counter and connect it to the variables over which to run the simulation. The primary part of the conversion process decomposes the algebraic expression into simple block operations and then translates simple block operations into a sequence of Ptolemy commands. Once the block operations have been converted into Ptolemy form, we add Ptolemy code to run the simulation.

The example to demonstrate the code generation ability is the two-channel non-uniform filter bank of Figure 5.6. Figure 7.2 draws the schematic of the filter bank, gives the *Mathematica* code representing the system, and adds the *Mathematica* commands necessary to invoke the simulation. In order for the simulation to run, we must specify the analysis/synthesis filters and the input signal $x[n]$. We use the filters designed in [82] so that the filter bank achieves near perfect reconstruction of

- Given
  - an algebraic expression parsed into tree form with variables as leaves and operators as nodes
  - a list of each {variable, minimum value, maximum value, increment} over which to run the simulation (the increment if not specified defaults to 1)
  - optionally, a list of values to assign to constants that appear in the algebraic expression (the assignment will take place in the generated Ptolemy code)
- Pre-process the algebraic expression by
  - converting LSI convolution operations to filtering operations
  - extracting constants from the algebraic expression (constants are symbols that the user has not specified as variables)
- Generate a complete Ptolemy simulation by
  - determining the Ptolemy code to use as the prolog
  - generating code that will allocate constant terms appearing in the expression, assigning to them the value given by the user (see above) or one if a value was not specified
  - generating code that will allocate variables
  - generating code that will define the algorithm by
    - converting the algebraic expression to block diagram form
    - converting the block diagram form to Ptolemy code
  - generating code to invoke the simulation

Figure 7.1: Algorithm to Convert Algebraic Expressions to Ptolemy Simulations



(a) Flow graph

```
upperchannel =
  Downsample[2,n][
    Convolve[n][g0[n], Upsample[3,n][
                         Downsample[3,n][
                           Convolve[n][h0[n],
                                       Upsample[2,n][x[n]]]]]]]
lowerchannel =
  Convolve[n][g1[n], Upsample[3,n][
                       Downsample[3,n][ Convolve[n][h1[n], x[n]]]]]
```

(b) Representation in the new environment

```
h0[n] = FIR[n, Hold[ReadList["ptolemy/h0", Number]]];
h1[n] = FIR[n, Hold[ReadList["ptolemy/h1", Number]]];

g0[n] = FIR[n, Hold[ReadList["ptolemy/g0", Number]]];
g1[n] = FIR[n, Hold[ReadList["ptolemy/g1", Number]]];

x[n] = Cos[2 Pi n / 3] Sinc[Pi n / 6] / 3;

PtolemyProgram[ upperchannel + lowerchannel, {n, 1, 100},
                Dialogue -> All ] >> "!interpreter"
```

(c) Specification of filters and input signal

Figure 7.2: Ptolemy Simulation of Filter Bank Run From Within *Mathematica*

the input signal. The filter coefficients will be read by Ptolemy when it runs the simulation (the `Hold` command prevents *Mathematica* from evaluating the `ReadList` command which would read the files in before code generation). We want to choose the input signal $x[n]$ to test the reconstruction properties of the filter bank. The filter bank decomposes the input frequency band into $(-\frac{2}{3}\pi, \frac{2}{3}\pi)$ for the upper channel and $(-\pi, -\frac{2}{3}\pi) \cup (\frac{2}{3}\pi, \pi)$ for the lower channel. So, one suitable choice for $x[n]$ is a bandpass signal having frequency content in the range of frequencies $(-\frac{5}{6}\pi, -\frac{1}{2}\pi) \cup (\frac{1}{2}\pi, \frac{5}{6}\pi)$. By setting the amplitude to be 1 over these bands, we can ask the MDSPPs to generate the time response of the bandpass signal. as shown in Figure 7.3. The intermediate block diagram form, which unravels the nested operations in the algorithm, is shown in Figure 7.4. The generated Ptolemy code is listed in Appendix A.1.

7.2.3 Code Generation After Algorithm Rearrangement

If the analysis/synthesis filters are implemented as either FIR or IIR filters, then the MDSPPs can find better implementations of the two-channel filter bank. Using the default cost function, which counts the number of additions, multiplication, and memory elements, a much better implementation results after rewriting each analysis and synthesis channel in its polyphase form:

```
upperchannel =
 PolyphaseResample[2, FIR[n, Hold[ReadList["ptolemy/g0",Number]]], 3, n][
  PolyphaseResample[3, FIR[n, Hold[ReadList["ptolemy/h0",Number]]], 2, n][
   x[n] ] ];

lowerchannel =
 PolyphaseUpsample[3, FIR[n, Hold[ReadList["ptolemy/g1",Number]]], n][
  PolyphaseDownsample[2, FIR[n, Hold[ReadList["ptolemy/h1",Number]]], n][
   x[n] ] ];
```

By specifing the same input `x[n]` as in Figure 7.2(c), the Ptolemy simulation is

```
In[14]:= freqResponse =
         CPulse[Pi/3, w + 5 Pi/6] + CPulse[Pi/3,  w - Pi/2]

                     -Pi                    5 Pi
Out[14]= CPulse    [--- + w] + CPulse    [---- + w]
               Pi/3  2               Pi/3  6

In[15]:= timeResponse = InvDTFTransform[ freqResponse, w, n ]

          (-2 I)/3 n Pi      n Pi     (2 I)/3 n Pi      n Pi
         E              Sinc[----]   E             Sinc[----]
                              6                           6
Out[15]= ------------------------ + ------------------------
                     6                           6

In[16]:= SPSimplify[ timeResponse, Variables -> n ]

             2 n Pi       n Pi
         Cos[------] Sinc[----]
               3           6
Out[16]= ----------------------
                   3
```

Figure 7.3: Deriving the Input Bandpass Signal for the Filter Bank Simulation

```
rationalconst1 := 1/3
timesconst1 := 2 Pi / 3
timesconst2 := Pi / 6
timesbyconstant1 := n timesconst1
cos1 := Cos[timesbyconstant1]
timesbyconstant2 := n timesconst2
sinc1 := Sinc[timesbyconstant2]
times1 := cos1 sinc1
timesbyconstant3 := rationalconst1 times1
upsample1 := Upsample   [timesbyconstant3]
                     2,n
fir1 := FIR[n, Hold[ReadList[ptolemy/h0, Number]]][upsample1]
downsample1 := Downsample   [fir1]
                          3,n
upsample2 := Upsample   [downsample1]
                     3,n
fir2 := FIR[n, Hold[ReadList[ptolemy/g0, Number]]][upsample2]
downsample2 := Downsample   [fir2]
                          2,n
fir3 := FIR[n, Hold[ReadList[ptolemy/h1, Number]]][
            timesbyconstant3]
downsample3 := Downsample   [fir3]
                          3,n
upsample3 := Upsample   [downsample3]
                     3,n
fir4 := FIR[n, Hold[ReadList[ptolemy/g1, Number]]][upsample3]
plus1 := downsample2 + fir4
```

Figure 7.4: Block Diagram Form of the Two-Channel Filter Bank

invoked by

```
PtolemyProgram[ upperchannel + lowerchannel, {n, 1, 100},
                Dialogue -> All ] >> "!interpreter"
```

The Ptolemy code for this more efficient filter bank structure is given in Appendix A.2.

7.3 Summary

This chapter demonstrates the ability of the MDSPPs to convert an algorithm composed of a set of mathematical formulas into either a set of equivalent lines of TEX code for typesetting or a complete Ptolemy program. Ptolemy program generation is necessary for the environment to take part in rapid prototyping because the algorithm can easily be embedded into a complex system. The complex system, once simulated in Ptolemy, can then be translated into a C program or DSP assembly language program. Until recently, Ptolemy has only offered limited abilities to process images [15]. Now, it supports multidimensional scheduling [16] so generating code for multidimensional multirate algorithms should be possible.

CHAPTER 8

Interactive Design of Two-Dimensional Decimation Systems

This chapter demonstrates the ability of the multidimensional signal processing packages (MDSPPs) to design a two-dimensional rational decimation system based on knowledge of the input signal's passband. The rational decimation system, which is drawn in Figure 8.1, resamples the bandlimited input signal at its Nyquist rate by expanding its frequency content to fill the entire fundamental frequency tile $\omega_1 \in [-\pi, \pi) \bigcup \omega_2 \in [-\pi, \pi)$. In this chapter, we represent the bandlimited frequency content as either a polygon or a rational resampling matrix. By representing the passband as a polygon, the user is free to define passbands with arbitrary shapes, either by sketching the shape on a graph with a mouse or by typing the coordinates of the vertices manually. In the alternate representation as a rational resampling matrix, the passband is formed by mapping the fundamental frequency tile by the inverse of the transpose of the rational resampling matrix. Under this mapping, the passband becomes a parallelogram whose vertices are rational multiples of π.

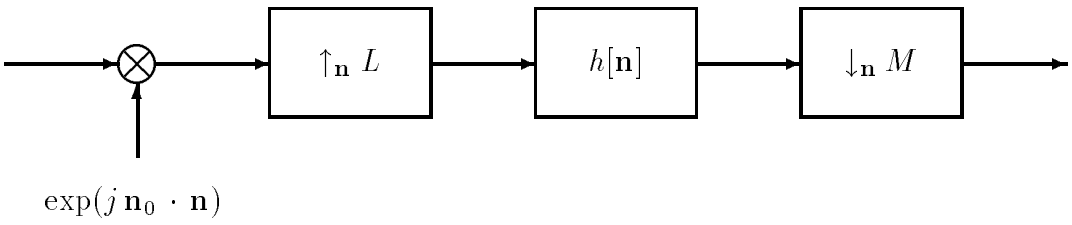


Figure 8.1: Flow Graph of a Two-Dimensional Decimator

Section 8.1 describes the theory underlying the design of two-dimensional decimators when given a passband of arbitrary shape. The design procedure requires four steps, each of which is realized by a routine in the MDSPPs. The four subroutines are combined into one `DesignDecimationSystem` command. Implementing the underlying theory requires a computing system that performs exact precision arithmetic and supports a polygon data structure. Section 8.2 gives three decimator design examples: a quincunx decimator, a decimator for a sketched passband, and a decimator for a circularly bandlimited signal.

8.1 Theory Underlying Decimator Design

This section discusses the theory behind the design of rational decimation systems. Rational decimation systems resample bandlimited input signals at their Nyquist rates. The extent of the frequency domain of the input signal is described either by a list of coordinates that form the vertices of a polygon or by a rational resampling matrix H. From a polygonal representation, the design procedure

1. computes the minimal rectangle with floating-point coordinates that circumscribes the passband,

2. finds the parallelogram whose coordinates are rational multiples of π, whose area is minimal, and whose extent includes the minimal rectangle,

3. shifts the center of the parallelogram to the origin,

4. defines the rational resampling matrix H that maps the parallelogram onto the fundamental frequency tile, and

5. factors the rational matrix H into two integer matrices L and M.

We implement Step 4 according to [83]. For Step 5, we factor the rational resampling matrix H by first decomposing H into its Smith-McMillan form and then collecting

terms:

$$H = U\,\Lambda\,V = U\,\Lambda_L^{-1}\,\Lambda_M\,V = \left(\Lambda_L\,U^{-1}\right)^{-1}\,(\Lambda_M V) = L^{-1}\,M \tag{8.1}$$

So, $L = \Lambda_L\,U^{-1}$ and $M = \Lambda_M V$. The diagonal elements of Λ_L are the inverse of the denominators of the diagonal elements of Λ, and the diagonal elements of Λ_M are the numerators of the diagonal elements of Λ. Since the rational numbers along the diagonal elements of Λ have already been reduced by the `SmithMcMillianForm` routine, Λ_L and Λ_M are relatively prime (on the right and the left), and therefore, so are L and M. Because L and M are relatively prime, polyphase implementations always exist for our rational decimator designs (see Figure 3.15 and [52]). Commutativity of L and M is not required for a polyphase implementation, but L and M can sometimes be adjusted so that their matrix products commute while maintaining their relative primeness. Letting $L = T\,\Lambda_L\,U^{-1}$ and $M = T\,\Lambda_M\,V$, the matrix products commute if $\Lambda_L\,V\,U = V\,U\,\Lambda_L$ then $T = U$ or if $\Lambda_M\,V\,U = V\,U\,\Lambda_M$ then $T = V^{-1}$. The rest of this section discusses the computations involved in Steps 1 and 2.

Step 1 of the design procedure finds the rectangle of minimal area that circumscribes the passband. For each polygon edge, the polygon is rotated so that the current edge would lie on the x-axis. The rectangle of smallest area that circumscribes the rotated polygon is then computed by finding the minimum and maximum coordinates of the rotated vertices. Once all E polygon edges have been processed, the rectangle with the smallest area is then chosen. Step 1 requires $\mathcal{O}(E^2)$ arithmetic operations and $\mathcal{O}(E^2)$ min and max operations.

From the rectangle computed in Step 1, Step 2 finds the parallelogram with minimal area that circumscribes the rectangle. Unlike the coordinates of the rectangle's vertices, each coordinate of the parallelogram's vertices must be a rational number times π. The procedure first sorts the vertices of the rectangle such that the first vertex is the upper left corner and the other vertices are in clockwise order. Then, the procedure rationalizes the division of three of the four rectangle coordinates with π (e.g. the coordinates of the upper left corner of the rectangle would decrease in the

x and increase in the y direction), and the fourth coordinate is computed from the other three so as to make sure that the bounding region is a parallelogram. Next, the procedure checks to make sure that all of the parallelogram vertices are outside, or on the boundaries of, the rectangle. If this condition does not hold true for all of the parallelogram vertices, then we shift the original parallelogram vertices in a clockwise direction and check again. If one or more parallelogram vertices is still inside the rectangle, then the parallelogram is shifted counter-clockwise. The procedure finishes when a valid parallelogram is found.

8.2 Design Examples

This section describes the automatic design of three different rational decimators that share the structure shown in Figure 8.1:

- a decimator for a diamond-shaped baseband (Figure 8.2),
- a decimator for an arbitrarily-shaped baseband (Figure 8.3), and
- a decimator for a circular baseband (Figure 8.4).

In all three cases, the baseband of the input signal is described by a polygon whose vertices were either sketched by placing points on the fundamental frequency tile with a mouse (as in Figures 8.2 and 8.3) or generated by a formula (as in Figure 8.4).

We have written a new routine called `DesignDecimationSystem` that designs a two-dimensional decimator given the passband represented by either a polygon or a resampling matrix. When given a polygon as its input, the `DesignDecimationSystem` routine first finds the rectangle with minimum area that circumscribes the polygon. Then, it computes the parallelogram that circumscribes the minimal rectangle yet has vertices that are rational numbers times π (the upper limit on the value of denominator of the rational numbers is controlled by the `Mod` option). Next, the `DesignDecimationSystem` routine shifts the center of the paralellogram to the origin,

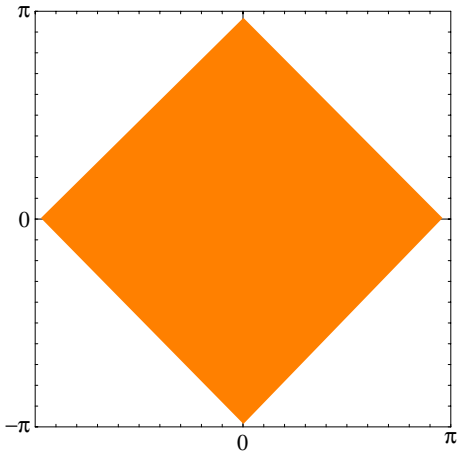
and the amount of the shift is the $\mathbf{n}_0$ vector in the modulator (see Figure 8.1). Based on the theorems in [83], `DesignDecimationSystem` then calculates the rational matrix that maps the shifted parallelogram onto the fundamental frequency tile, i.e. maps the parallelogram's vertices to the vertices $\{(-\pi, \pi), (\pi, \pi), (\pi, -\pi), (-\pi, -\pi)\}$. The rational matrix is then factored into $L^{-1}M$ which gives the upsampling matrix L and the downsampling matrix M.

Each figure in this section contains the output of three *Mathematica* commands. The first two define and plot the baseband of the input signal as a polygon. The third command, a call to `DesignDecimationSystem`, produces two plots: the baseband superimposed on the minimal rectangle bounding the baseband (in the upper right portion of the figure) and the baseband superimposed on the parallelogram bounding the minimal rectangle (in the lower right portion of the figure). In these plots, the baseband is shaded grey, and the visible portions of the bounding rectangle and parallelogram are shown in black. The `DesignDecimationSystem` routine reports the packing efficiency and the input-output compression ratio obtained by the decimation system. The compression ratio is defined as the area of the bounding parallelogram divided by the area of the fundamental frequency tile ($4\pi^2$). The `DesignDecimationSystem` routine returns the parameters of the rational decimation system: the modulation shift $\mathbf{n}_0$, the upsampling matrix L, and the downsampling matrix M.

Figure 8.2 illustrates the design of a rational decimation system for a sketched passband that is nearly quincunx. The fitted rectangle in the final plot is a quincunx passband. Since the upsampling matrix has a determinant of one, it serves only to rotate the passband to the proper shape so that the downsampler will map the passband onto the entire fundamental frequency tile.

Figure 8.3 shows the automatic design of a rational decimation system for an arbitrarily-sketched baseband. In this case, the bounding rectangle and the bounding parallelogram actually extend beyond the fundamental frequency tile. Therefore, the

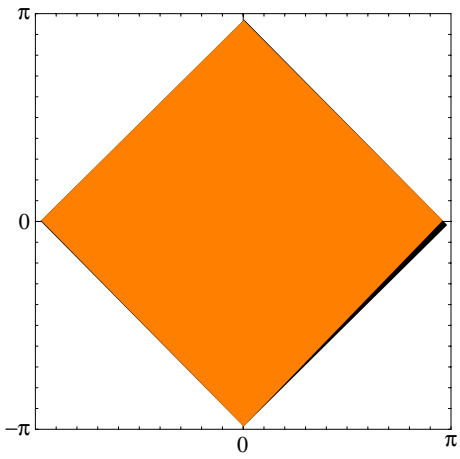
```
poly = Polygon[
(* paste points below and evaluate expression *)
{{0.006425, 3.042108}, {3.016461, 0.015067},
{0.006425, -3.097004}, {-3.054628, 0.015067}}
];

Show [ Graphics[ { RGBColor[1,1/2,0], poly } ],
       AspectRatio -> 1, Axes -> True,
       Frame -> True,
       FrameTicks -> { piTicks, piTicks },
       PlotRange -> {{-Pi, Pi}, {-Pi, Pi}} ]
```



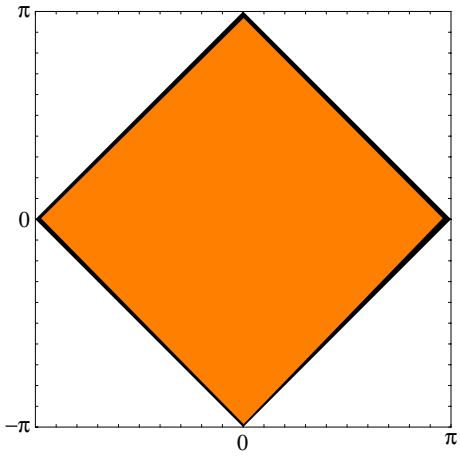

```
-Graphics-
```

This example finds the two-dimensional rational decimator to resample a diamond-shaped passband at its Nyquist rate. The four vertices of the polygon defining passband were chosen graphically so the passband is not exactly a rotated square. From the polygon, `DesignDecimationSystem` computes the modulator shift $\mathbf{n}_0$ and the up/downsampling matrices L and M for the decimator structure shown in the previous figure. This decimation system achieves a compression ratio of 2-to-1 ($|\det M|/|\det L| = 2$). Since `DesignDecimationSystem` always renders the circumscribing rectangles behind the baseband, the rectangles appear here as black outlines because they fit the passband so closely.

```
{ shift, upMatrix, downMatrix } =
  DesignDecimationSystem[
    poly, Dialogue -> All, Mod -> 10 ]
Best packing efficiency with rotated rectangle
   having real-valued coordinates: 98.6%
```




```
Actual packing efficiency: 94.4%
  (out of a best possible 98.6%)
```




```
The compression ratio is 2-to-1.
{{0, 0}, {{1, 0}, {1, 1}}, {{1, 1}, {0, 2}}}
```

$$n_0 = (0, 0)$$

$$L = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$$M = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}$$

Figure 8.2: Automatic Design of a Quincunx Decimator

frequency content of these bounding regions wraps around to lower frequencies. The original coordinates of the parallelogram are used in the computation of the shift vector and the rational sampling matrix $L^{-1}M$.

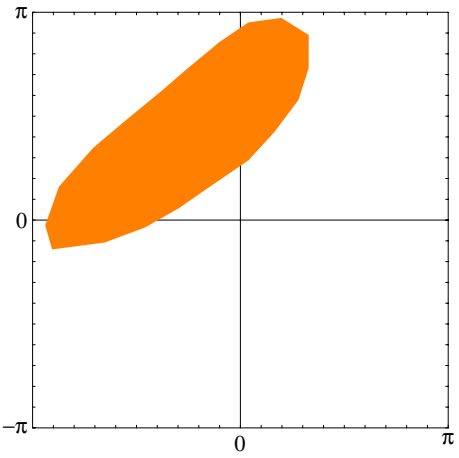
Figure 8.4 demonstrates the ability of our method to achieve good compression for circularly bandlimited signals. Slightly better compression could be achieved by circumscribing the circular passband with a regular hexagon. For the regular hexagon, the maximum packing efficiency is 86.6% for a hexagonal cell [53], compared to a maximum packing efficiency of 78.5% for our method. Note that the packing efficiency for the decimator in Figure 8.4 is higher than 78.5% because the input baseband is a polygon that approximates a circular passband.

8.3 Summary

This chapter demonstrates the automatic design of the components of two-dimensional decimation systems (having the structure shown in Figure 8.1) to resample a bandlimited input signal at its Nyquist rate. For the design procedure, the passband of the input signal is described as a polygon. From the vertices of polygon, the procedure first finds the rectangle with minimum area that circumscribes the polygon. Then, it computes the parallelogram whose vertices are pairs of rational numbers times π and whose extent includes the minimal rectangle. Next, the procedure shifts the center of the paralellogram to the origin to find $\mathbf{n}_0$. For the next step, the procedure uses the ideas in [83] to map the shifted parallelogram onto the fundamental frequency tile by means of a rational matrix. Finally, we factor the rational matrix into its Smith-McMillan form so that we can choose L and M such that $L^{-1}M$ equals the resampling matrix. Because of the way in which we use the Smith-McMillan form, L and M are always relatively prime on the left (see Section 3.1.5) so efficient polyphase forms for our decimator designs always exist.

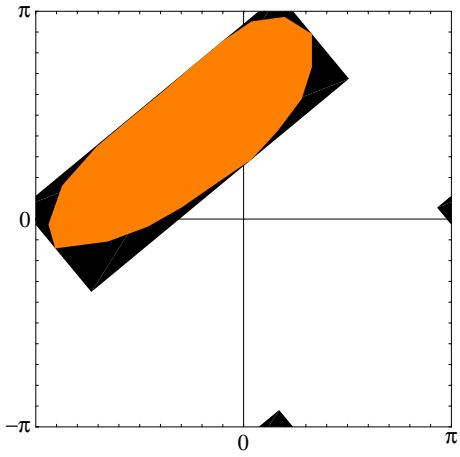
We have implemented the design procedure in the `DesignDecimationSystem`

```
poly = Polygon[
(* paste points below and evaluate expression *)
{{-2.846744, -0.443421}, {-2.950194, -0.081346},
{-2.743294, 0.50487}, {-2.226045, 1.091086},
{-1.777762, 1.470402}, {-1.191546, 1.953168},
{-0.81223, 2.280759}, {-0.312222, 2.694558},
{0.118819, 2.987666}, {0.618826, 3.056633},
{1.032626, 2.798008}, {1.032626, 2.298001},
{0.877451, 1.815234}, {0.515377, 1.332469},
{0.118819, 0.901428}, {-0.536364, 0.453145},
{-0.932921, 0.177279}, {-1.450171, -0.115829},
{-2.053628, -0.339971}}
];

Show [ Graphics[ { RGBColor[1,1/2,0], poly } ],
       AspectRatio -> 1, Axes -> True,
       Frame -> True,
       FrameTicks -> { piTicks, piTicks },
       PlotRange -> {{-Pi, Pi}, {-Pi, Pi}} ]
```

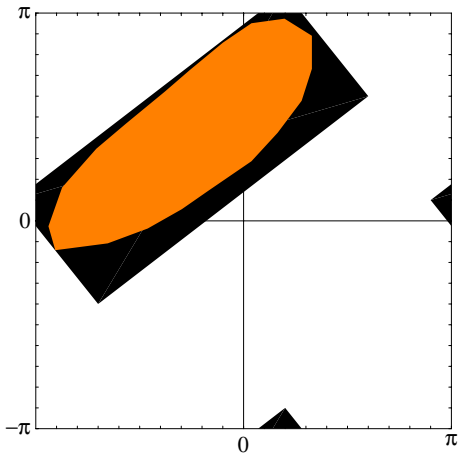


-Graphics-

```
{ shift, upMatrix, downMatrix } =
  DesignDecimationSystem[
    poly, Dialogue -> All, Mod -> 10 ]
Best packing efficiency with rotated rectangle
   having real-valued coordinates: 78.7%
```




```
Actual packing efficiency: 63.6%
   (out of a best possible 78.7%)
```




```
The compression ratio is 80-to-21.

{{-Pi/4, 7 Pi/20}, {{21, -21}, {-4, 5}},
  {{92, 4}, {-20, 0}}}
```

This example finds the two-dimensional rational decimation system to resample the passband shown directly above at its Nyquist rate. The parts of the circumscribing rectangle and parallelogram that extend outside of the fundamental frequency tile wrap around because the tile is periodic with period 2π in each frequency variable. This rational decimation system achieves a compression ratio of about 4-to-1 ($\frac{|\det M|}{|\det L|} = \frac{80}{21} \approx 4$).

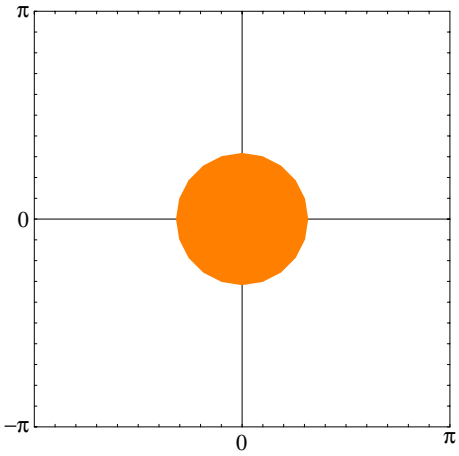
$$n_0 = (\frac{-\pi}{4}, \frac{7\pi}{20})$$

$$L = \begin{bmatrix} 21 & -21 \\ -4 & 5 \end{bmatrix}$$

$$M = \begin{bmatrix} 92 & 4 \\ -20 & 0 \end{bmatrix}$$

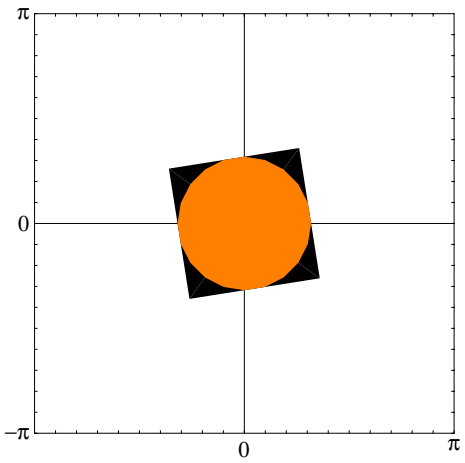
Figure 8.3: Automatic Design of a Decimator for an Arbitrarily-Shaped Passband

```
poly =
  Polygon[
    N[ Table[ { Cos[theta], Sin[theta] },
              { theta, Pi/10, 2 Pi, Pi/10 } ] ]
];

Show [ Graphics[ { RGBColor[1,1/2,0], poly } ],
       AspectRatio -> 1, Axes -> True,
       Frame -> True,
       FrameTicks -> { piTicks, piTicks },
       PlotRange -> {{-Pi, Pi}, {-Pi, Pi}} ]
```

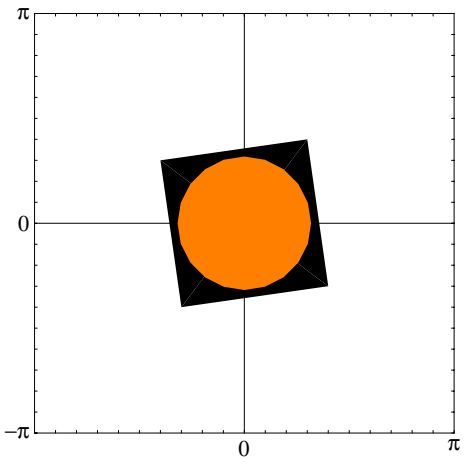



```
-Graphics-
```

```
{ shiftVector, upMatrix, downMatrix } =
  DesignDecimationSystem[
    poly, Dialogue -> All, Mod -> 10 ]
Best packing efficiency with rotated rectangle
   having real-valued coordinates: 79.2%
```




```
Actual packing efficiency: 62.6%
  (out of a best possible 79.2%)
```




```
The compression ratio is 8-to-1.
{{0, 0}, {{5, 0}, {7, 1}}, {{2, 14}, {0, 20}}}
```

This example finds the two-dimensional rational decimation system to resample a circular passband (with radius 1) near its Nyquist rate. The rational decimator design achieves an 8-to-1 compression ratio ($|\det M|/|\det L| = 8$). The theoretical upper limit on the compression ratio is the ratio of the area of the fundamental frequency tile ($4\pi^2$) to the area of the circle (π) which is 4π (approximately 12.5-to-1.).

$$n_0 = (0,0)$$

$$L = \begin{bmatrix} 5 & 0 \\ 7 & 1 \end{bmatrix}$$

$$M = \begin{bmatrix} 2 & 14 \\ 0 & 20 \end{bmatrix}$$

Figure 8.4: Automatic Design of a Decimator for Circularly Bandlimited Signals

routine. The *Mathematica* Notebook interface supports the graphical interaction necessary for the user to sketch the passband of the input signal. We utilize the exact precision arithmetic operations in *Mathematica* within the `DesignDecimationSystem` routine. We chose not to implement the filter design algorithms in *Mathematica* that would have enabled us to design the decimation filter. In any case, the decimation filter has a passband that is a symmetric parallelopiped (i.e. symmetric about the origin) whose vertices are $\pi L^{-1} M v_i$ for $i = 1 \ldots 4$ where v is the set of vertices $\{(-1,1), (1,1), (1,-1), (-1,-1)\}$ which forms a square [83].

Armed with the ability to design a rational decimator based on a given passband in the frequency domain, a bank of rational decimators can be designed to resample a union of possibly disconnected passbands. For each pair of passbands, the intersection of the parallelograms bounding the passbands would have to be taken into account. A conceptually simple way to take the overlap into account is to remove the intersecting region from one of the parallelograms and then design the two decimators. A better way is to use the overlapping region to generate a third decimator to account for the difference between the frequency content of the the decimator pair and the frequency content of the input signal. The outputs of two decimators can only be added if the two $L^{-1}M$ terms are equal (i.e. they operate at the output sampling rates). A bank of rational decimators that cover the fundamental frequency tile without overlapping is better known as a critically sampled filter bank [56, 84].

CHAPTER 9

Impact on the Engineering Curriculum

The previous four chapters discuss the use of the MDSPPs to analyze, design, and prototype signal processing algorithms. This chapter reports how the analysis capabilities of the MDSPPs coupled with electronic documents written in *Mathematica*'s Notebook interface have helped students learn linear systems theory [23, 24, 25].

Computer algebra (CA) systems are well-suited for solving problems that can be reduced to a set of mathematical formulas. CA environments enable students to check answers to problems and create their own example problems. These systems can also be programmed to give the intermediate steps leading to a solution.

In order to solve problems rapidly with a CA program, students must use a mixture of computer and mathematics skills. These skills can develop naturally during an undergraduate education if CA environments are introduced early in the curriculum. First-year students at several schools are already using these environments to learn calculus. Putting CA systems in the classroom will require a restructuring of the class to integrate computer assignments into the everyday activities of the students. Professors may also need to make sure that students will not replace thinking and learning with a dependence on symbolic calculators, much as the professors already do with respect to pocket calculators.

CA environments are useful to engineering students because CA environments are proficient at factoring polynomials and performing partial fraction decompositions. Their algebraic abilities enable students to perform node and mesh analysis of simple resistive networks [85] and RLC circuits [86]. Because of their abilities to manipulate complex-valued numbers and functions, they can compute phasors for AC circuits, and their calculus abilities can help students in analyzing and simulating

passive and active electrical networks [87]. Those CA environments that can solve simultaneous non-linear equations are potentially useful in designing active circuits [88, 89].

A computer algebra program must be customized in order to integrate it into a signals and systems curriculum. A successful sequence of signals and systems courses would teach students to grasp

1. the representation of signals as complex numbers, matrices, vectors and polynomials,
2. the representation of systems as mathematical operators that map signals into other signals,
3. new functions such as step, impulse and sinc functions,
4. new operators such as shifters, modulators and filters,
5. new symbolic mathematical operations of linear transforms and linear convolution defined for signals, and
6. new graphical representations such as pole-zero diagrams and frequency response plots.

CA systems typically address the first topic above because they support a variety of mathematical structures and routines that work on them. These programs also support most of the continuous functions commonly used in defining signals (trigonometric, Bessel, exponential, and so forth), but they generally do not understand sinc functions or piecewise continuous functions such as steps and pulses. Therefore, their linear transforms do not recognize typical signal processing expressions. Furthermore, their transforms cannot show the student how to compute the transform by hand, and their Laplace transforms (and z-transforms if implemented) do not track the region of convergence. They do not provide routines for linear convolution. For use in

graduate signal processing classes, CA environments are further limited because their linear transforms are typically unilateral and one-dimensional. Lastly, CA programs provide few of the common graphical representations of signals.

By developing the multidimensional signal processing packages (MDSPPs) introduced in Chapter 4, we have customized the CA program *Mathematica* to support many aspects of a signals and systems curriculum. The MDSPPs define signals and systems missing in the core of *Mathematica*, as shown earlier in Tables 4.1 and 4.2. Based on this representation of signals and systems, the MDSPPs implement convolution, transforms, signal plotting, signal analysis, and other high-level abilities as listed in Table 5.2. The plotting routines can help students visualize signals and systems, whereas the other routines can show the student answers to convolution and transform problems by hand.

The rest of this chapter discusses the use of the Notebook interface to *Mathematica* in helping students learn linear systems theory. In Section 2.3, the Notebook interface was described as a multimedia interface to the *Mathematica* kernel. By adding formatted text, graphics, and sound to examples written as *Mathematica* commands, Notebooks can become living interactive documents. We have written Notebooks to

- introduce students to the extended *Mathematica* environment (Section 9.1),
- teach students key topics in a tutorial format (Section 9.2),
- provide on-line documentation (Section 9.3), and
- empower students to evaluate their own knowledge (Section 9.4)

Section 9.5 evaluates these efforts at integrating the multidimensional signal processing packages and Notebooks into classes in different disciplines at different universities.

9.1 Student Handout

The student handout was developed to be a user's guide for the juniors and seniors enrolled in the two required signals and systems courses at Georgia Tech. The courses use [5] as the primary text. The student handout supplements the text as it shows students how to use MDSPPs to solve homework problems from the text.

The student handout has undergone three iterations. The first section introduces the student to *Mathematica* and the MDSPPs. The next two sections discuss how students in the first and second courses would use the MDSPPs to tackle homework problems. The remaining sections work an example problem in detail, give hints for commonly encountered errors, and summarize the important signals and systems defined by the MDSPPs. The handout is a collection of six *Mathematica* Notebooks.

9.2 Interactive Tutorial Notebooks

Using the MDSPPs, Notebooks can provide examples of algebraic forms of signal processing expressions for the students to evaluate. By adding text, equations, and graphics explaining these examples, Notebooks can take the form of interactive tutorials. Our tutorial Notebooks arrange information in a hierarchy of sections, subsections, and so forth. When opening one of the tutorial Notebooks, a student would see the table of contents. Students can open a section of interest to delve deeper into a specific topic, or they can skim the Notebook sequentially to get an overview. Students do have the option of searching for keywords (highlighted by the author) and patterns (chosen by the student).

To date, we have written the following tutorials:

AnalogFilters shows how to design Bessel and classical analog filters: Butterworth, Chebyshev, and Elliptic.

DTFT teaches the discrete-time Fourier transform (DTFT) by relating it to the discrete Fourier transform (DFT), Fourier series, and the continuous-time Fourier transform.

PiecewiseConvolution discusses both discrete and piecewise continuous convolution and demonstrates the mechanics of the "flip-and-slide" approach to convolution by means of animation.

zTransformI defines the bilateral z-transform and introduces the region of convergence (ROC) which is necessary for uniqueness; discussing the ROC leads naturally into the subject of stability and causality of a signal based on the location of its poles relative to the ROC.

zTransformII demonstrates relationships between the z-transform and the DTFT.

zTransformIII shows how to choose poles and zeros to construct digital filters.

Each tutorial contains enough formatted text and equations to introduce the subject. Their purpose, however, is not to duplicate material already found in textbooks but instead to enable students to interact with the subject material. Students can explore the subject by evaluating code, viewing animations, or listening to sound. As an area of future work, we hope to write tutorial Notebooks on a comprehensive range of linear systems topics as shown in Table 9.1.

All of the tutorial Notebooks listed above contain animation. In the Notebook on analog filter design, for example, the student can visualize the changes in the filter's magnitude response as one design parameter varies. The Notebook provides three animations of magnitude responses— one for a lowpass Butterworth filter of varying order, one for a bandpass elliptic filter with varying ripple and fixed order, and one for bandpass elliptic filter with varying ripple and order.

The Notebook on piecewise convolution shows how to break down a general convolution problem into several smaller convolutions. It also helps the student identify overlapping intervals by proceeding step-by-step through a simple problem. To

Topic	Notebook That Covers It
Continuous Convolution	`PiecewiseConvolution`
Laplace transforms	
Solving Differential Equations	
Analog Filter Design	`AnalogFilters`
Continuous Fourier Transforms	
Communication Schemes	
Discrete convolution	`PiecewiseConvolution`
z-Transforms	`zTransformI-II`
Solving Difference Equations	
Digital Filter Design	`zTransformIII`
Discrete Fourier Transforms	`DTFT`
Classical Feedback Control	*
State-Space Methods	*

* These topics are covered by a set of Notebooks developed at the Case Western Reserve University to accompany a controls textbook [90]. The implementation of control theory relies heavily on the signal processing packages.

Table 9.1: Coverage of Linear Systems Topics by Tutorial Notebooks

reinforce the procedure, an animation sequence illustrates the convolution of a truncated ramp $x(t)$ and a pulse $h(t)$ [3, 26]. In the animation sequence, the first frame superimposes the functions. The sixth frame shows $h(t-\tau)$ for $t = 0$ which is $h(\tau)$ "flipped" about the τ axis. A side benefit of the sixth frame is that it separates the two functions being convolved. The remaining frames illustrate the sliding of $h(t-\tau)$ across $x(\tau)$ for different values of t. The final frame gives the result of the convolution. When viewing the animation, *Mathematica* allows the student to control the frame rate, freeze the sequence, etc. The student can visualize the important intervals over which to integrate and observe that the maximum value of the output function occurs when there is maximum overlap between the two functions being convolved.

The z-transform Notebooks contain many animations. Several relate pole-zero diagrams to magnitude frequency responses— a basic idea behind digital filter design [75, 91]. Two animations illustrate the effect on the magnitude response of a fourth-order IIR elliptic filter when two of its poles are slightly perturbed (see Figure 9.1). Not only do these animations reinforce the degradation that can result when filter coefficients are quantized, they also show that the poles closest to the unit circle are the most sensitive to perturbations (such as quantization).

9.3 Notebooks Serving as On-Line Reference

Accompanying the tutorial Notebooks is a set of Notebooks that provide on-line help and on-line reference.

EducationalTool describes uses of the multidimensional signal processing packages and Notebooks in colleges and universities with many examples for the student to evaluate; it is a multimedia version of [23].

README briefly introduces the multidimensional signal processing packages and Notebooks.

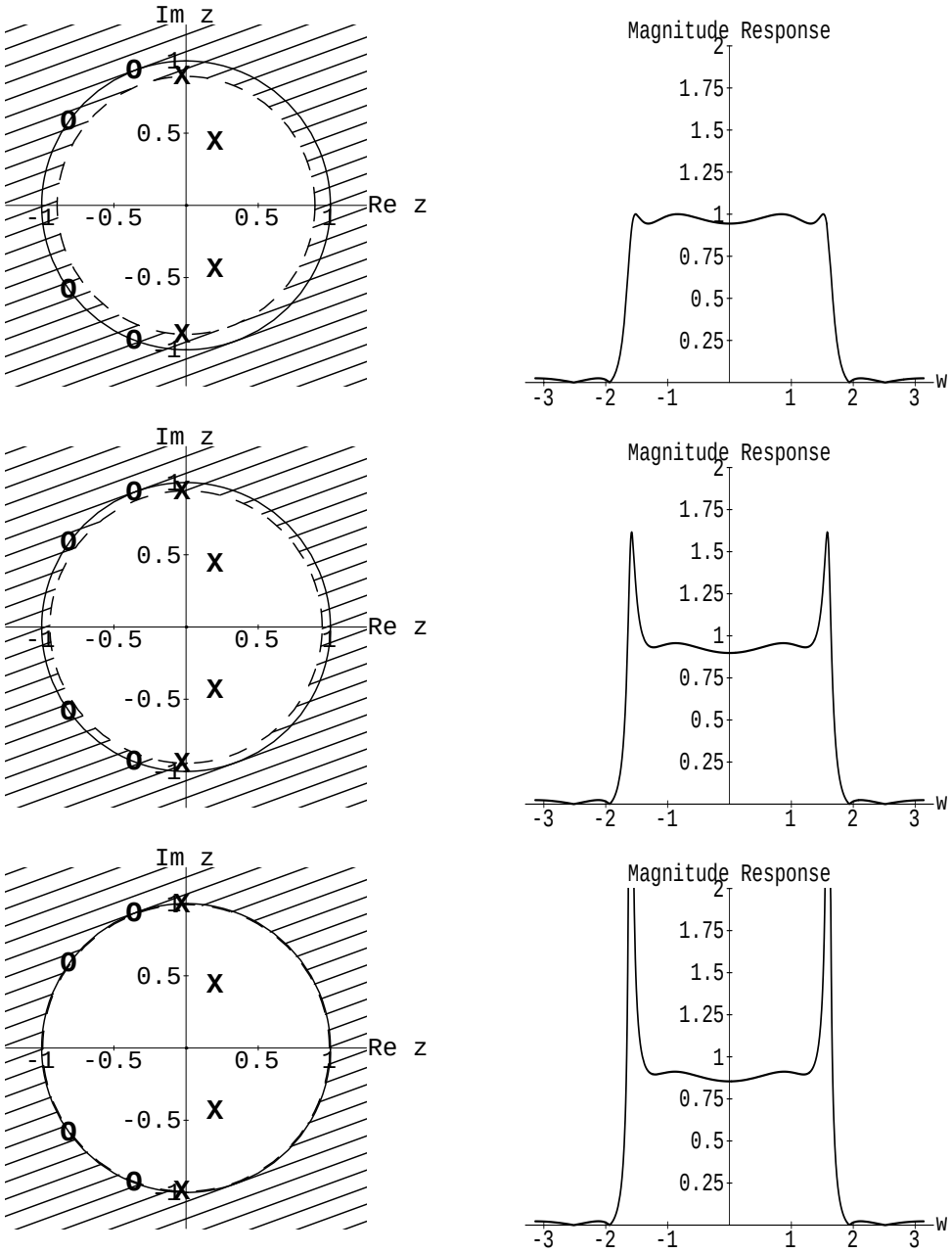


Figure 9.1: Animation of Filter Response for Corrupted Poles
Selected frames in an animation showing how a perturbation in the radius of the outer pole pair corrupts the magnitude response near the band edge

SignalProcessingExamples gives an overview of the signal processing packages with many examples for the student to evaluate; it is a multimedia version of [2], but it also contains a history of the different versions of and bug fixes to the MDSPPs.

SignalProcessingIntroduction introduces *Mathematica*, signal processing, and the signal processing packages.

SignalProcessingUsage provides complete on-line reference manual for every new routine, function, and so forth defined by the signal processing packages

The **README** and **SignalProcessingIntroduction** Notebooks were written to initiate new users. **SignalProcessingExamples** and **EducationalTool** provide overviews of the signal processing packages and Notebooks, respectively. **SignalProcessingUsage** provides an on-line reference to all of the new functions, data structures, and operations.

9.4 Notebooks for Self-Evaluation

In a standard text, students have access to a limited number of example and homework problems. Using the Notebook format, professors can write their own example problems for students to study or complete.

In the Notebook, the instructor can pose questions that require intensive symbolic and numeric computations. Students answer the questions with graphs, text, and *Mathematica* code. Once completed, the student could submit the ASCII Notebook by electronic mail for grading, as is done in some calculus courses [92]. Such Notebooks could help students learn how to generate their own problems and solutions in *Mathematica*.

At North Carolina State University, electrical engineering students are asked to complete two exercise Notebooks which use the interactive graphics capabilities of

the Notebook interface. The following Notebooks, which do not rely on any external packages, guide students through exercises in order to help them evaluate their knowledge of a topic: [25]

makeconv asks the student to sketch a convolution kernel with lowpass characteristics which the Notebook checks. The question is repeated for a highpass kernel. Students can convolve various signals with their own kernels.

polezero asks the student to design four different digital filters by pole-zero placement (a sixth-order all-pole bandpass, an all-pole low pass, a better low pass, and a notch filter). Students evaluate their choices by inspecting the magnitude response. At any time, students can have *Mathematica* check the pole-zero placement.

9.5 Summary

Computer algebra environments are currently being used in the teaching of mathematics, physics, and engineering. This chapter discusses the impact of the multidimensional signal processing packages and Notebooks for *Mathematica* on the engineering curriculum. The extended *Mathematica* environment is useful in learning linear systems theory, especially in the areas of communications, controls, and signal processing. In the MDSPPs, students explore linear systems theory by

- studying the steps the environment takes in solving homework problems,
- interacting with the multimedia Notebook tutorials, and
- evaluating their own knowledge though iterative dialogue with prepared Notebooks.

Students are using the multidimensional signal processing packages and Notebooks in engineering and mathematics classes at several colleges and universities as

shown in Table 9.5. For example, graduate Civil Engineering students at the Pennsylvania State University have been using MDSPPs since 1991 to explore topics related to water quality. In the class, convolution arises in studying rainfall runoff in watersheds. Using a simple exponential response function that models absorption by the soil, the students have been able to see the impact of an arbitrary rain event shape on runoff. Animating this convolution has helped them visualize the principle of causal systems, where the present runoff is a composite of past conditions.

Based on the signal processing packages, Narasingarao Sreenath at the Case Western Reserve University has developed a symbolic controls systems analysis package (COSYPAK). Over eighty percent of COSYPAK consists of a subset of the MDSPPs. COSYPAK supports Notebooks that supplement the material in the textbook [90] used by students in two graduate controls courses (see Table 9.5). He first presented the extended system to the students as an optional part of the courses. However, since few students wanted to take the time to learn a new environment, the professor now requires the students to solve homework problems with it.

In 1989, the Georgia Institute of Technology first introduced *Mathematica* into a graduate mathematics course teaching mathematical programming. Since then, the School of Mathematics has used *Mathematica* in teaching advanced engineering mathematics and transform theory. Beginning in the Fall of 1991, the mathematics department required students in several calculus sections to use *Mathematica* to solve, verify, and document three projects per quarter. One quarter later, the School of Electrical Engineering began using *Mathematica* in two introductory signals and systems courses. In these classes, the students complete homework assignments involving convolution and transforms using our signal processing extensions. Students can work on their own computer at home or on a Macintosh, NeXT, or Sun computer on campus. Students who enjoy working with computers benefit the most. On the other hand, those students who have to learn a new computer, networks, and *Mathematica* all at once encounter much difficulty. These difficulties will decrease

School	Course	Title	Uses Extensions for
Case Western Reserve Univ.	ESYS 304	Control Engineering I	Laplace transforms and plotting
	ESYS 306	Control Engineering II	As above
Georgia Institute of Technology	EE 3216	Circuits, Signals, and Systems I	Convolution and Fourier transforms
	EE 3221	Circuits, Signals, and Systems II	DTFT and z-transform
	EE 4078	Digital Signal Processing	z-transforms
	MATH 4581	Advanced Engineering Mathematics I	Laplace transforms
	MATH 8xxx	Mathematical Programming	Examples of programming
Penn. State University	CE 597	Tracer and Contaminant Transport in Groundwater Systems	Convolution and solving Linear con. coeff. differential equ's
Rose-Hulman Institute of Technology	MA 201	Differential Equations I	Laplace transforms Linear con. coeff. differential equ's
	MA 202	Differential Equations II	Linear con. coeff. difference equ's
	EE 391	Intro. to Communication Systems	Fourier transforms
Stanford University	EE 265	Applications of the Short-Time Fourier Transform	Fourier transforms and sequence plots
University of the Pacific	EE 121	Systems I	Convolution, Laplace, Fourier
	EE 122	Systems II	Convolution, z-transform
Univ. of Penn.	EE 314	Signals and Linear Systems	Computer projects
Washington State University	EE 321	Electrical Circuits II	Laplace transforms
	EE 341	Communications Systems	Convolution and Fourier transforms
	EE 464	Digital Signal Processing	Convolution and z-transforms
	EE 582	Multirate Signal Processing	Filter design system analysis

Table 9.2: Past and Present Uses of the Signal Processing Extensions

as the new breed of calculus students enter the electrical engineering program. The *Mathematica* initiatives in calculus and in the signals and systems courses, as well as the use of other computer algebra programs in electrical engineering courses, have set the stage for further integration of *Mathematica* into the electrical engineering curriculum.

CHAPTER 10

Conclusion

Kopec has advocated a broader view of signals as objects having their own properties (slots) and procedures (methods). He originally implemented his ideas as the interpretive Signal Representation Language (SRL) [28, 29, 30]. Descendants of SRL, as shown in Figure 10.1, have taken one of three independent directions: simulation, knowledge-based signal processing (KBSP), and algorithm design. Two prominent algorithm design environments, SPLICE [7, 8, 9] and ADE [10, 11, 12], express one-dimensional (multirate) systems as nested formulas. On the other hand, the Blosim [33], Gabriel [34], and Ptolemy [13] simulation environments represent systems as interconnected hierarchical block diagrams. This thesis discusses a new algorithm design environment, called the multidimensional signal processing packages (MDSPPs), which automates the design and analysis of multidimensional multirate systems expressed as algebraic formulas and converts the algebraic representation into block diagrams for inclusion in Ptolemy.

The algorithm design environments of SPLICE, ADE, and MetaMorph [31] are implemented in Lisp. SPLICE and ADE are unavailable for general use because they only run under Symbolics dialects of Lisp. Furthermore, these environments are missing detailed mathematical knowledge of signals and systems, extensive graphics capabilities, and the ability to translate algorithms into other environments.

This thesis describes a different approach. Instead of extending Lisp, we have started at a much higher level by using a symbolic mathematics system, *Mathematica*. With *Mathematica*, we automatically gain the ability to define objects, match patterns, and express conditional rules (see Section 2.3). *Mathematica* manipulates formulas and supports a programming language. In *Mathematica*'s programming

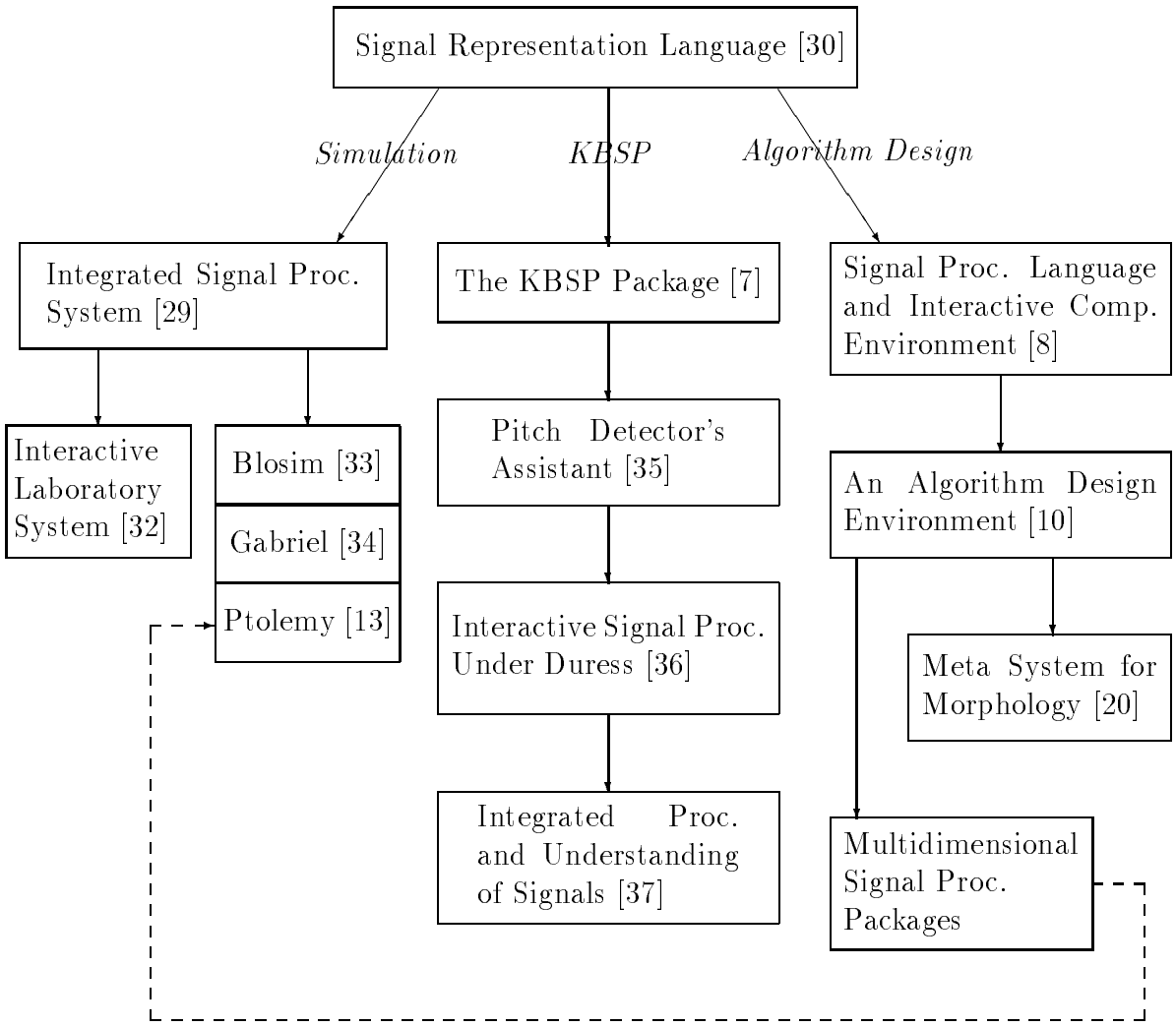


Figure 10.1: Descendants of Kopec's Signal Representation Language
Updated version of Figure 2.2 to include the MDSPPs.
The dashed line indicates code generation.

language, we have developed a set of multidimensional signal processing packages (MDSPPs) that represent signals as functions and systems as operators (Chapter 4). Signal processing algorithms are expressed as formulas containing signals and systems. Based on this algebraic representation, the MDSPPs can analyze, simplify, rearrange, and optimize algorithms (Chapters 5 and 6). Once an algorithm has been developed, it can be transported directly into a block diagram environment, either to be simulated or inserted into a more complex design (Chapter 7).

We have written several *Mathematica* multimedia Notebooks to accompany the MDSPPs so that the extensions are self-documenting. By combining features of the *Mathematica* kernel with the Notebook interface, we can convert between numeric, symbolic, and graphical representations of formulas. We utilize this combination to automate the design of two-dimensional rational decimation systems (Chapter 8) and to turn *Mathematica* into an educational tool for signal processing (Chapter 9).

10.1 Contributions

The contributions of this thesis, as shown in Table 10.1, relate to the abilities of our MDSPPs for *Mathematica*. The MDSPPs have the ingredients necessary to be simultaneously useful to designers implementing signal processing algorithms, students learning signals and systems, and researchers developing new signal processing theory. All three groups can benefit from its symbolic and graphical analyses of one-dimensional and multidimensional (multirate) systems.

Designers can use MDSPPs to help decide on values of free parameters and to find better implementations of algorithms. Once an algorithm has been designed, the MDSPPs can convert it into a working Ptolemy simulation under a synchronous data flow model (i.e. uniform sampling). Ptolemy can generate complete C and DSP assembly language programs as a part of a rapid prototyping process.

Students can learn the theoretical operations underlying signal processing by

Theory	• Generalized Rules for Multidimensional Multirate Systems
	• Efficient Algorithms to Implement the Rules
	• New Smith Form Decomposition Routines
Analysis	• Multidimensional Discrete-Time and Continuous-Time Transforms
	• Multidimensional Discrete-Time and Continuous-Time Stability Analysis Based on Z and Laplace Transforms
	• Properties of Multidimensional Signals and Systems
	• Visualization of Resampling in Two Dimensions
Algorithms	• Automatic Rearrangement of Multidimensional Multirate Systems
	• Cost Functions for Multidimensional Algorithms
Prototyping	• Automatic Code Generation For Algorithms
	• Block Diagram Representations for Algebraic Representations of Algorithms
Design	• Interactive Design of Two-Dimensional Decimation Systems
	• Automatic Conversion of Graphical Specifications into Usable Design Specifications
Education	• Textual and Graphical Justification of Answers
	• Exploration of Linear Systems Theory
	• Student Self-Evaluation

Table 10.1: Contributions of the Thesis

interacting with the MDSPPs and the accompanying electronic Notebooks. For both digital and analog signals, the packages can animate the flip-and-slide approach to piecewise convolution. The multidimensional linear transforms, the differential and difference equation solvers, and many other routines in the MDSPPs can show their intermediate calculations in the form of natural language dialogue. The MDSPPs also provide many of the graphical representations of one-dimensional and two-dimensional systems such as lollipop plots for one-dimensional sequences and pole-zero root maps for two-dimensional systems. By interacting with the tutorial Notebooks, students can learn difficult concepts such as convolution and the z-transform. A different set of Notebooks helps students evaluate their own knowledge of convolution and digital filter design. Other Notebooks form a user's guide and serve as on-line help.

Researchers can benefit from the rich set of symbolic analyses on signal processing algorithms. But by itself, *Mathematica* can empower researchers to develop new algorithms. It has helped us derive number theoretic algorithms that impose different kinds of structure on Smith form decompositions of integer and rational matrices [27] which have aided in the design of non-uniform multidimensional filter banks [73].

Besides discussing the new environment defined by the MDSPPs, the thesis also formalizes the rules governing the interaction of linear shift-invariant and periodically time-varying systems. The thesis also identifies efficient algorithms to implement these rules. In determining the commutativity of an upsampler and downsampler in cascade, we developed two equivalent pairs of conditions, one based on the time domain and one based on the frequency domain. After analysis of the underlying algorithms, we determined that the conditions based on the time domain should be used to check for commutativity because the underlying algorithms are faster.

The thesis also extends many of the one-dimensional abilities of E-SPLICE and ADE to multiple dimensions. We have identified many of the important multidimensional signal and system properties for multiple dimensions. In addition, we have developed cost functions to rank equivalent multidimensional multirate algorithms.

10.2 Future Research

The MDSPPs turn a symbolic mathematics environment into a viable signal processing research, design, and educational tool. The tool manipulates signals and systems as algebraic expressions, including automatic rearrangement and simplification of expressions. This ability to rearrange components of expressions is crucial in finding better implementations of algorithms, but many unresolved issues remain. Table 10.2 lists avenues of future research, which we will highlight next.

One open issue is to find the best way to constrain the space of equivalent forms of an algorithm. Another open issue is defining appropriate cost functions for multidimensional multirate systems. The current cost function measures the number of additions, multiplications, and memory elements, but it neglects to count the number of memory accesses and to take into account delay in communicating data between components in an algorithm.

The environment does not stand alone but instead generates TEX code and complete simulations for Ptolemy. Now that Ptolemy supports multidimensional scheduling, it should be possible to generate working Ptolemy simulations for multidimensional multirate structures such as non-uniform multidimensional filter banks. It should also be possible to read a Ptolemy description into *Mathematica*, especially one that was originally generated by the MDSPPs. This ability would allow feedback of lessons learned while simulating the algorithm to an earlier stage in the prototyping process. Future work may extend the MDSPPs to convert algebraic expressions to other target environments such as Matlab and the Signal Processing WorkSystem^{TM} [93]. Furthermore, the MDSPPs could be ported to other computer algebra platforms such as Maple to make the MDSPPs even more widely available.

Please direct any electronic correspondence concerning this research to Brian Evans at `evans@eedsp.gatech.edu`. At the time of writing, a freely distributable version of the multidimensional signal processing packages and Notebooks is available via anonymous FTP to `gauss.eedsp.gatech.edu` (IP #130.207.226.24).

Theory	• Effect of Different Smith Forms on Algorithms
Analysis	• True Multidimensional Transforms
	• Inequality Reasoning To Support Stability Analysis
	• Visualization of Resampling in Three Dimensions
Algorithms	• Better Heuristics To Rearrange Algorithms
	• Better Cost Functions
Prototyping	• Feedback From Ptolemy After Code Generation
Design	• Interactive Decimator Design in Three Dimensions
Education	• Methods for Student Self-Evaluation On More Topics
	• Automated Evaluation of a Student's Abilities

Table 10.2: Future Research

APPENDIX A

Ptolemy Simulation Code

This appendix lists the Ptolemy source code generated for the non-uniform two-channel filter bank shown in Figure 7.2 by the `PtolemyProgram` command in the MDSPPs. The generated source code has the following structure: header, constants, variables, the algorithm(s), and simulation directives. In this example, the source code has been generated for Version 0.3.1 of Ptolemy. This version of the Ptolemy interpreter uses a Lisp-like syntax. The number sign `#` represents a comment.

A.1 Code Generation for the Two-Channel Non-Uniform Filter Bank

This is the code generated by the MDSPPs for the two-channel non-uniform filter bank.

```
(load "~/mathematica/SignalProcessing/ObjectOriented/header.pt")

#
# Constants
#


#
# Variables
#

(star n FloatRamp)
(setstate n value "1")
(node n)
```

```
(nodeconnect (n output) n)


#
# Algorithm
#

# 1/3
(state rationalconst1 float "0.3333333333333333")

# (2*Pi)/3
(state timesconst1 float "2.094395102393195")

# Pi/6
(state timesconst2 float "0.5235987755982988")

(star timesbyconstant1 FloatGain)
(setstate timesbyconstant1 gain "timesconst1")
(node timesbyconstant1)
(nodeconnect (timesbyconstant1 output) timesbyconstant1)

(nodeconnect (timesbyconstant1 input) n)

(star cos1 Cos)
(node cos1)
(nodeconnect (cos1 output) cos1)

(nodeconnect (cos1 input) timesbyconstant1)

(star timesbyconstant2 FloatGain)
(setstate timesbyconstant2 gain "timesconst2")
(node timesbyconstant2)
(nodeconnect (timesbyconstant2 output) timesbyconstant2)

(nodeconnect (timesbyconstant2 input) n)

(star sinc1 Sinc)
(node sinc1)
(nodeconnect (sinc1 output) sinc1)

(nodeconnect (sinc1 input) timesbyconstant2)
```

```
(star times1 FloatProduct)
(node times1)
(nodeconnect (times1 output) times1)
(numports times1 input 2)

(nodeconnect (times1 "input#1") cos1)
(nodeconnect (times1 "input#2") sinc1)

(star timesbyconstant3 FloatGain)
(setstate timesbyconstant3 gain "rationalconst1")
(node timesbyconstant3)
(nodeconnect (timesbyconstant3 output) timesbyconstant3)

(nodeconnect (timesbyconstant3 input) times1)

(star upsample1 UpSample)
(setstate upsample1 factor 2)
(node upsample1)
(nodeconnect (upsample1 output) upsample1)

(nodeconnect (upsample1 input) timesbyconstant3)

(star fir1 FIR)
(setstate fir1 taps "<ptolemy/h0")
(node fir1)
(nodeconnect (fir1 signalOut) fir1)

(nodeconnect (fir1 signalIn) upsample1)

(star downsample1 DownSample)
(setstate downsample1 factor 3)
(node downsample1)
(nodeconnect (downsample1 output) downsample1)

(nodeconnect (downsample1 input) fir1)

(star upsample2 UpSample)
(setstate upsample2 factor 3)
(node upsample2)
(nodeconnect (upsample2 output) upsample2)

(nodeconnect (upsample2 input) downsample1)
```

```
(star fir2 FIR)
(setstate fir2 taps "<ptolemy/g0")
(node fir2)
(nodeconnect (fir2 signalOut) fir2)

(nodeconnect (fir2 signalIn) upsample2)

(star downsample2 DownSample)
(setstate downsample2 factor 2)
(node downsample2)
(nodeconnect (downsample2 output) downsample2)

(nodeconnect (downsample2 input) fir2)

(star fir3 FIR)
(setstate fir3 taps "<ptolemy/h1")
(node fir3)
(nodeconnect (fir3 signalOut) fir3)

(nodeconnect (fir3 signalIn) timesbyconstant3)

(star downsample3 DownSample)
(setstate downsample3 factor 3)
(node downsample3)
(nodeconnect (downsample3 output) downsample3)

(nodeconnect (downsample3 input) fir3)

(star upsample3 UpSample)
(setstate upsample3 factor 3)
(node upsample3)
(nodeconnect (upsample3 output) upsample3)

(nodeconnect (upsample3 input) downsample3)

(star fir4 FIR)
(setstate fir4 taps "<ptolemy/g1")
(node fir4)
(nodeconnect (fir4 signalOut) fir4)

(nodeconnect (fir4 signalIn) upsample3)
```

```
(star plus1 FloatSum)
(node plus1)
(nodeconnect (plus1 output) plus1)
(numports plus1 input 2)

(nodeconnect (plus1 "input#1") downsample2)
(nodeconnect (plus1 "input#2") fir4)

#
# Run Simulation
#

(star xygraph1 XYgraph)
(nodeconnect (xygraph1 xInput) n)

(nodeconnect (xygraph1 input) plus1)

(run 100)
(wrapup)
```

A.2 Code Generation for a More Efficient Form of the Filter Bank

This is the code generated after the MDSPPs rearrange the two-channel non-uniform filter bank to find the polyphase implementation for each analysis and synthesis branch.

```
(load "~/mathematica/SignalProcessing/ObjectOriented/header.pt")


#
# Constants
#
```

```
#
# Variables
#

(star n FloatRamp)
(setstate n value "1")
(node n)
(nodeconnect (n output) n)


#
# Algorithm
#

# 1/3
(state rationalconst1 float "0.3333333333333333")

# (2*Pi)/3
(state timesconst1 float "2.094395102393195")

# Pi/6
(state timesconst2 float "0.5235987755982988")

(star timesbyconstant1 FloatGain)
(setstate timesbyconstant1 gain "timesconst1")
(node timesbyconstant1)
(nodeconnect (timesbyconstant1 output) timesbyconstant1)

(nodeconnect (timesbyconstant1 input) n)

(star cos1 Cos)
(node cos1)
(nodeconnect (cos1 output) cos1)

(nodeconnect (cos1 input) timesbyconstant1)

(star timesbyconstant2 FloatGain)
(setstate timesbyconstant2 gain "timesconst2")
(node timesbyconstant2)
(nodeconnect (timesbyconstant2 output) timesbyconstant2)

(nodeconnect (timesbyconstant2 input) n)
```

```
(star sinc1 Sinc)
(node sinc1)
(nodeconnect (sinc1 output) sinc1)

(nodeconnect (sinc1 input) timesbyconstant2)

(star times1 FloatProduct)
(node times1)
(nodeconnect (times1 output) times1)
(numports times1 input 2)

(nodeconnect (times1 "input#1") cos1)
(nodeconnect (times1 "input#2") sinc1)

(star timesbyconstant3 FloatGain)
(setstate timesbyconstant3 gain "rationalconst1")
(node timesbyconstant3)
(nodeconnect (timesbyconstant3 output) timesbyconstant3)

(nodeconnect (timesbyconstant3 input) times1)

(star polyphaseresample1 FIR)
(setstate polyphaseresample1 interpolation 3)
(setstate polyphaseresample1 taps ""<ptolemy/h0"")
(setstate polyphaseresample1 decimation 2)
(node polyphaseresample1)
(nodeconnect (polyphaseresample1 output) polyphaseresample1)

(nodeconnect (polyphaseresample1 input) timesbyconstant3)

(star polyphaseresample2 FIR)
(setstate polyphaseresample2 interpolation 2)
(setstate polyphaseresample2 taps ""<ptolemy/g0"")
(setstate polyphaseresample2 decimation 3)
(node polyphaseresample2)
(nodeconnect (polyphaseresample2 output) polyphaseresample2)

(nodeconnect (polyphaseresample2 input) polyphaseresample1)

(star polyphasedownsample1 FIR)
(setstate polyphasedownsample1 decimation 2)
```

```
(setstate polyphasedownsample1 taps ""<ptolemy/h1"")
(node polyphasedownsample1)
(nodeconnect (polyphasedownsample1 output) polyphasedownsample1)

(nodeconnect (polyphasedownsample1 input) timesbyconstant3)

(star polyphaseupsample1 FIR)
(setstate polyphaseupsample1 decimation 3)
(setstate polyphaseupsample1 taps ""<ptolemy/g1"")
(node polyphaseupsample1)
(nodeconnect (polyphaseupsample1 output) polyphaseupsample1)

(nodeconnect (polyphaseupsample1 input) polyphasedownsample1)

(star plus1 FloatSum)
(node plus1)
(nodeconnect (plus1 output) plus1)
(numports plus1 input 2)

(nodeconnect (plus1 "input#1") polyphaseresample2)
(nodeconnect (plus1 "input#2") polyphaseupsample1)


#
# Run Simulation
#

(star xygraph1 XYgraph)
(nodeconnect (xygraph1 xInput) n)

(nodeconnect (xygraph1 input) plus1)

(run 100)
(wrapup)
```

APPENDIX B

Glossary

ADE Algorithm Design Environment by Covell which is a Meta-System† for one-dimensional multirate signal processing algorithms

CAD Computer-Aided Design

Conditional rule an *if...then* rule with a condition and a consequence, e.g. if $a < b$ and $b < c$ then $a < c$

DFT Discrete Fourier Transform (discrete-time to discrete-frequency transform)

DSP Digital Signal Processing

DTFT Discrete-Time Fourier Transform (discrete-time to continuous-frequency transform)

Equivalence space the collection of all expressions equivalent to a certain expression; the space can often be arranged in tree structure in which each child node is different from its parent by one rearrangement† rule

E-SPLICE Extended SPLICE†

FIR Finite Impulse Response (filter)

FPD Fundamental Parallelepiped

Fundamental frequency tile since the discrete-time frequency domain is periodic with period 2π in each of the discrete-time frequency variables, one legitimate choice of the fundamental frequency tile is $\omega_i \in [-\pi, \pi)$ in each discrete-time frequency variable ω_i

IIR Infinite Impulse Response (filter)

ILS Interactive Laboratory System

IPUS Integrated Processing and Understanding of Signals

ISP Integrated Signal Processing System

ISPUD Interactive Signal Processing Under Duress developed for the design and analysis of multi-microphone hearing aids (actually, "Under Duress" is an artificial appendage to make the acronym pronounceable)

KBSP Knowledge-Based Signal Processing. The KBSP Package was a software tool written by W. Dove, C. Myers, and E. Milios at MIT.

Linear A system h is linear if for any constants a and b, $h\{a\,x(t) + b\,y(t)\} = a\,h\{x(t)\} + b\,h\{y(t)\}$. Equivalently, a system h is linear if for any constant c, $h\{c\,x(t)\} = c\,x(t)$ and $h\{x(t) + y(t)\} = h\{x(t)\} + h\{y(t)\}$.

LPTV Linear† Periodically Time-Varying† describes a periodic system that is time-varying over each period

LSI Linear† Shift-Invariant†

LTV Linear† Time-Varying†

MDSPPs Multidimensional Signal Processing Packages for *Mathematica*

MetaMorph Meta-System† for morphological signals (sets) and systems (set operations), i.e. a class of non-linear systems

Meta-Systems Symbolic reasoning systems, i.e. symbolic analysis and manipulation of elements in a domain of knowledge

PDA Pitch Detector's Assistant

Ptolemy Algorithm simulation tool which can generate source and VHDL† code for algorithms

Quincunx A quincunx resampling matrix is a family of resampling matrices with determinant of 2 or -2 and entries that are ± 1. The lattice (sampling grid) generated by these matrices correspond to diagonal interleaving commonly found in video signals.

Rearrangement rule a conditional† rule that rewrites a set of algebraic operations by rearranging the order of operations or by replacing the operations with an equivalent set of operations, e.g. switching the order of FIR and IIR filtering.

Regular unimodular A unimodular matrix has a determinant of -1 or 1. Resampling a discrete-time signal by a regular unimodular integer matrix only shuffles the signal— no samples are deleted or inserted.

Resampling matrix a square non-singular integer matrix

Rewrite rule a rearrangement† or simplification† rule

ROC The Region of Convergence of either the z or Laplace transform of a signal. It represents the range of complex values of the transform variable(s) for which the transform is valid.

Rule Base A collection of related conditional† rules

Shift-Invariant A system is shift-invariant if letting $h\{x(t)\} = y(t)$, $h\{x(t-t_0)\} = y(t-t_0)$ for all t_0.

Simplification rule a conditional† rule that removes redundant operations in an expression, e.g. modulating then demodulating by using the same modulation function or upsampling then downsampling by the same resampling matrix.

Smith form The Smith form of an $m \times n$ non-singular integer matrix is $U\Lambda V$ where the $m \times m$ U integer matrix and the $n \times n$ V integer matrix are regular unimodular† and the $m \times n$ Λ matrix is diagonal. It is the analog to singular value decomposition (SVD) over the semigroup of integer matrices. Like SVD, the Smith form decomposition makes multivariate operations (like up/downsampling) separable.

Smith-McMillan form The Smith-McMillan form is the extension of the Smith form† to non-singular rational matrices. The only difference is that Λ is a rational diagonal matrix.

SPLICE Signal Processing Language and Interactive Computing Environment, a forerunner of ADE†

SRL Signal Representation Language

Time-Varying not Shift-Invariant†

Unimodular A unimodular matrix has a determinant of -1, 0, or 1.

VHDL VLSI† Hardware Descriptive Language which allows the reuse of VLSI† designs for different VLSI† technologies

VLSI Very Large Scale Integration of microelectronic circuits

Bibliography

[1] B. L. Evans, J. H. McClellan, and W. B. McClure, "Symbolic z-transforms using DSP knowledge bases," in *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, (Albuquerque, NM), pp. 1775–1778, Apr. 1990.

[2] B. L. Evans, J. H. McClellan, and W. B. McClure, "Symbolic transforms with applications to signal processing," *The Mathematica Journal*, vol. 1, no. 2, pp. 70–80, Fall, 1990.

[3] B. L. Evans and J. H. McClellan, "Symbolic analysis of signals and systems," in *Symbolic and Knowledge-Based Signal Processing* (A. Oppenheim and H. Nawab, eds.), pp. 88–141, Englewood Cliffs, NJ: Prentice-Hall, 1992.

[4] S. Wolfram, *Mathematica: A System for Doing Mathematics by Computer*. Redwood City, CA: Addison-Wesley, second ed., 1991.

[5] A. V. Oppenheim and A. Willsky, *Signals and Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1983.

[6] A. Guessoum, *Fast Algorithms for the Multidimensional Discrete Fourier Transform*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, June 1984.

[7] W. Dove, C. Myers, and E. Milios, "An object-oriented signal processing environment: The Knowledge-Based Signal Processing Package," Tech. Rep. 502, MIT Research Laboratory for Electronics, Cambridge, MA, 1984.

[8] C. S. Myers, *Signal Representation for Symbolic and Numeric Processing*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, Aug. 1986. Research Laboratory for Electronics Tech. Rep. 521.

[9] C. Myers, "Symbolic representation and manipulation of signals," in *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, (Dallas, TX), pp. 2400–2403, Apr. 1987.

[10] M. M. Covell, *An Algorithm Design Environment for Signal Processing*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, Dec. 1989. Research Laboratory for Electronics Tech. Rep. 549.

[11] M. M. Covell, "An algorithm design environment for signal processing," *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, pp. 1779–1782, Apr. 1990.

[12] M. M. Covell and J. Richardson, "A new, efficient structure for the short-time Fourier transform with an application in code-division sonar imaging," *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, pp. 2041–2044, May 1991.

[13] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A platform for heterogeneous simulation and prototyping," in *Proc. of the 1991 European Simulation Conf.*, (Copenhagen, Denmark), July 1991.

[14] J. Buck, S. Ha, E. Lee, and D. Messerschmitt, "Multirate signal processing in Ptolemy," in *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, (Toronto, Canada), pp. 1245–1248, Mar. 1991.

[15] J. Buck, E. Goei, S. Ha, I. Kuroda, P. Lapsley, E. Lee, and D. Messerschmitt, "The Almagest: Manual for Ptolemy," Tech. Rep. for Ptolemy 0.3.1, The University of California at Berkeley, Berkeley, CA, 1991.

[16] E. A. Lee, "Representing and exploiting data parallelism using multidimensional dataflow diagrams," in *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, vol. I, (Minneapolis, MN), pp. 453–456, Apr. 1993.

[17] W. Dove and C. Myers, "Rapid prototyping of application specific signal processors," Tech. Rep. 1–2300, Lockheed Sanders Inc., 95 Canal Street, Nashua, NH, 1992. Sponsored by DARPA Contact MDA972-92-C-0058.

[18] C. H. Richardson and R. W. Schafer, "Symbolic manipulation and analysis of morphological algorithms," in *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, (Albuquerque, NM), pp. 2173–2176, Apr. 1990.

[19] C. H. Richardson and R. W. Schafer, "An environment for the automatic manipulation and analysis of morphological algorithms," in *Proc. of SPIE, Image Algebra, and Morphological Image Processing*, (Albuquerque, NM), pp. 262–273, July 1990.

[20] C. H. Richardson, *The Symbolic Representation, Analysis, and Manipulation of Morphological Algorithms*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, Dec. 1991.

[21] C. H. Richardson and R. W. Schafer, "The representation of morphological systems and meta-systems for automatic symbolic manipulations," in *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, (Minneapolis, MN), Apr. 1993.

[22] M. Slaney, "Interactive signal processing documents," *IEEE ASSP Magazine*, vol. 7, pp. 8–20, Apr. 1990.

[23] B. L. Evans, J. H. McClellan, and K. A. West, "Mathematica as an educational tool for signal processing," in *Proc. IEEE Southeastern Conf.*, (Williamsburg, VA), pp. 1175–1179, Apr. 1991.

[24] B. L. Evans, L. J. Karam, K. A. West, and J. H. McClellan, "Learning signals and systems with Mathematica," *IEEE Trans. on Education*, vol. 36, pp. 72–78, Feb. 1993.

[25] B. L. Evans, J. H. McClellan, and H. J. Trussell, "Investigating signal processing theory with Mathematica," in *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, vol. I, (Minneapolis, MN), pp. 12–15, Apr. 1993.

[26] K. A. West and J. H. McClellan, "Symbolic convolution," *IEEE Trans. on Education*, 1993. To Be Published.

[27] B. L. Evans, T. R. Gardos, and J. H. McClellan, "Imposing structure on Smith form decompositions of rational resampling matrices," *IEEE Trans. on Signal Processing*, vol. ASSP-42, Apr. 1994.

[28] G. Kopec, *The Representation of Discrete-Time Signals and Systems in Programs*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1980.

[29] G. Kopec, "The Integrated Signal Processing System ISP," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-32, Aug. 1984.

[30] G. Kopec, "The Signal Representation Language SRL," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-33, pp. 921–932, Aug. 1985.

[31] A. V. Oppenheim and S. H. Nawab, eds., *Symbolic and Knowledge-Based Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1992.

[32] *The Interactive Laboratory System (ILS)*. Signal Technology Inc., Goleta, CA, 1989.

[33] D. Hait, "The BLOSIM simulation program," Master's thesis, The University of California at Berkeley, Berkeley, CA, Nov. 1985.

[34] E. A. Lee, W. H. Ho, E. Goei, J. Bier, and S. Bhattacharyya, "Gabriel: A design environment for DSP," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-37, pp. 1751–1762, Nov. 1989.

[35] W. Dove, "Knowledge-based pitch detection," Tech. Rep. 518, MIT Research Laboratory for Electronics, Cambridge, MA, June 1986.

[36] P. Peterson and J. Frisbie, "Interactive environment for signal processing on a VAX computer," in *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, (Dallas, TX), pp. 1891–1893, Apr. 1987.

[37] H. Nawab and V. Lesser, "Integrated processing and understanding of signals," in *Symbolic and Knowledge-Based Signal Processing* (A. Oppenheim and H. Nawab, eds.), pp. 251–285, Englewood Cliffs, NJ: Prentice-Hall, 1992.

[38] H. Abelson and G. Sussman, *Structure and Interpretation of Computer Programs*. Cambridge, MA: MIT Press, 1985.

[39] T. Kailath, *Linear Systems*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1980.

[40] *N!Power*. Signal Technology Inc., Goleta, CA, 1990.

[41] M. M. Covell. Personal Correspondence, May 1993.

[42] B. Char and et al., *Maple Reference Manual*. Waterloo, Canada: WATCOM Publications, 1988.

[43] R. J. Fateman, "A review of MACSYMA," *IEEE Trans. on Knowledge and Data Eng.*, vol. 1, pp. 133–145, Mar. 1989.

[44] J. Nielsen, *HyperText & HyperMedia*. San Diego, CA: Academic Press, Inc., 1990.

[45] R. E. Maeder, *Programming in Mathematica.* Redwood City, CA: Addison-Wesley, 2 ed., 1991.

[46] N. Blachman, *Mathematica: A Practical Approach.* Englewood Cliffs, NJ: Prentice-Hall, 1992.

[47] R. E. Maeder, *Programming in Mathematica.* Redwood City, CA: Addison-Wesley, 1989.

[48] P. Winston and B. Horn, eds., *LISP.* Reading, MA: Addison-Wesley, 3 ed., 1989.

[49] R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing.* Englewood Cliffs, NJ: Prentice-Hall, 1983.

[50] P. P. Vaidyanathan, "Multirate digital filters, filter banks, polyphase networks, and applications: A tutorial," *Proc. of the IEEE*, vol. 78, pp. 56–93, Jan. 1990.

[51] T. Chen and P. P. Vaidyanathan, "Commutativity of D-dimensional decimation and expansion matrices, and applications to rational decimation systems," in *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, vol. 4, (San Francisco, CA), pp. 637–640, Mar. 1992.

[52] T. Chen and P. P. Vaidyanathan, "The role of integer matrices in multidimensional multirate systems," *IEEE Trans. on Signal Processing*, vol. ASSP-41, pp. 1035–1047, Mar. 1993.

[53] D. E. Dudgeon and R. M. Mersereau, *Multidimensional Digital Signal Processing.* Englewood Cliffs, NJ: Prentice-Hall, 1984.

[54] D. Petersen and D. Middleton, "Sampling and reconstruction of wave-number limited functions in N-dimensional Euclidean spaces," *Information and Control*, vol. 5, pp. 279–323, 1962.

[55] E. Dubois, "The sampling and reconstruction of time-varying imagery with application in video systems," *Proc. of the IEEE*, vol. 73, pp. 502–522, Apr. 1985.

[56] R. H. Bamberger, *The Directional Filter Bank: A Multirate Filter Bank for the Directional Decomposition of Images*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, Dec. 1990.

[57] J. Cassels, *An Introduction to the Geometry of Numbers*. Berlin, Germany: Springer-Verlag, 1959.

[58] G. D. Forney, "Coset codes — Part I: Introduction and geometrical classification," *IEEE Transactions on Information Theory*, vol. 34, pp. 1123–1151, Sept. 1988.

[59] E. Viscito and J. Allebach, "Design of perfect reconstruction multidimensional filter banks using cascaded Smith form matrices," in *Proc. IEEE Int. Sym. Circuits and Systems*, (Espoo, Finland), pp. 831–834, June 1988.

[60] A. Kaufmann and A. Henry-Labordère, *Integer and Mixed Programming: Theory and Applications*. New York: Academic Press, 1977.

[61] P. P. Vaidyanathan, "The role of Smith-Form decomposition of integer matrices in multidimensional multirate systems," in *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, (Toronto, Canada), pp. 1777–1780, May 1991.

[62] C. Iliopoulos, "Worst-case complexity bounds on algorithms for computing the canonical structure of finite Abelian groups and the Hermite and Smith normal forms," *SIAM Journal on Computing*, vol. 18, pp. 658–669, 1989.

[63] C. C. MacDuffee, *The Theory of Matrices*. Berlin, Germany: Springer-Verlag, 1933.

[64] P. P. Vaidyanathan, "Quadrature mirror filter banks, M-band extensions and perfect reconstruction techniques," *IEEE ASSP Magazine*, vol. 4, pp. 4–20, July 1987.

[65] M. Vetterli, "A theory of multirate filter banks," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-35, pp. 356–372, Mar. 1987.

[66] K. Nayebi, *A Time Domain Framework for the Analysis and Design of FIR Multirate Filter Bank Systems.* PhD thesis, Georgia Institute of Technology, Atlanta, GA, Dec. 1990.

[67] E. Viscito and J. Allebach, "The analysis and design of multidimensional FIR perfect reconstruction filter banks for arbitrary sampling lattices," *IEEE Trans. on Circuits and Systems*, vol. CAS-38, pp. 29–41, Jan. 1991.

[68] T. R. Gardos. Personal Correspondence, Feb. 1992.

[69] G. Havas, "Coset enumeration strategies," in *Proc. Int. Sym. on Symbolic and Algebraic Computation*, (Bonn, Germany), pp. 191–199, July 1991.

[70] J. Kovacevic and M. Vetterli, "The commutativity of up/downsampling in two dimensions," Tech. Rep. 186-90-16, Center for Telecommunications Research at Columbia University, New York, NY, 1990.

[71] B. L. Evans, J. H. McClellan, and R. H. Bamberger, "A symbolic algebra for linear multidimensional multirate systems," in *Proc. Conf. on Inf. Sciences and Systems*, (Princeton, NJ), pp. 387–393, Mar. 1992.

[72] B. L. Evans and J. H. McClellan, "Rules for multidimensional multirate structures," *IEEE Trans. on Signal Processing*, vol. ASSP-42, Apr. 1994.

[73] T. R. Gardos, K. Nayebi, and R. M. Mersereau, "Analysis and design of multidimensional, non-uniform band filter banks," in *SPIE Proc. Visual Communications and Image Processing*, pp. 49–60, Nov. 1992.

[74] R. N. Bracewell, *The Fourier Transform and Its Applications.* New York: McGraw-Hill, second ed., 1986.

[75] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing.* Englewood Cliffs, NJ: Prentice-Hall, 1989.

[76] Z. Zuhao, "The square matrix rule of the convolution integral," *IEEE Trans. on Education*, pp. 369–372, Nov. 1990.

[77] M. M. Covell, C. S. Myers, and A. V. Oppenheim, "Computer-aided algorithm design and rearrangement," in *Symbolic and Knowledge-Based Signal Processing* (A. Oppenheim and H. Nawab, eds.), Englewood Cliffs, NJ: Prentice-Hall, 1992.

[78] D. E. Knuth, *The TeXbook.* Reading, MA: Addison-Wesley, 1984.

[79] E. Kant, F. Daube, W. MacGregor, and J. Wald, "Automated synthesis of finite difference programs," in *Symbolic Computations and Their Impact on Mechanics*, vol. 205, (Dallas, TX), pp. 45–61, 1990. Presented at the Winter Annual Meeting of the American Society of Mechanical Engineers.

[80] E. Kant, F. Daube, W. MacGregor, J. Wald, E. Houstis, J. Rice, and R. Vichnevetsky, "Knowledge-based program generation for mathematical modeling," in *Proc. of the Second IMACS International Conference on Expert Systems for Numerical Computing*, (West Lafayette, IN), pp. 371–92, Apr. 1992.

[81] E. Kant, "Synthesis of mathematical-modeling software," *IEEE Software*, vol. 10, pp. 30–41, May 1993.

[82] K. Nayebi, T. Barnwell, and M. J. T. Smith, "Nonuniform filter banks: A reconstruction and design theory," *IEEE Trans. on Signal Processing*, vol. ASSP-41, pp. 1114–1127, Mar. 1993.

[83] T. Chen and P. P. Vaidyanathan, "On the choice of rational decimation systems for multidimensional signals," in *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Processing*, vol. V, (Minneapolis, MN), pp. 527–530, Apr. 1993.

[84] T. R. Gardos, *Analysis and Design of Multidimensional FIR Filter Banks*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, June 1993.

[85] M. Yoder, "The use of symbolic algebra in electrical engineering," *Computers in Education Division of ASEE*, vol. 1, pp. 56–60, 1991.

[86] J. Vlach, "Network theory and CAD," *IEEE Trans. on Education*, vol. 36, pp. 23–27, Feb. 1993.

[87] A. Riddle, "A nodal circuit analysis program," *The Mathematica Journal*, vol. 1, pp. 62–68, Summer 1990.

[88] J. Vlach and K. Singhal, eds., *Computer Methods for Circuit Analysis and Design*. New York: Van Nostrand Reinhold, 1983.

[89] J. Vlach and K. Singhal, eds., *Selected Papers on Computer-Aided Design of Analog Networks*. New York: IEEE Press, 1987.

[90] K. Ogata, *Modern Control Engineering*. Englewood Cliffs, NJ: Prentice-Hall, second ed., 1991.

[91] T. W. Parks and C. S. Burrus, *Digital Filter Design*. New York: John Wiley and Sons, 1987.

[92] D. Brown, H. Porta, and J. Uhl, "Calculus and Mathematica: Courseware for the nineties," *The Mathematica Journal*, vol. 1, pp. 43–50, Summer 1990.

[93] *Signal Processing WorkSystem*. Comdisco Systems, Inc., Foster City, CA.

Vita

Brian "Ignatius" Lawrence Evans was born in Cincinnati, Ohio, on Mother's Day, May 10, 1965. He does not remember why he went to the Rose-Hulman Institute of Technology, an all-male college whose name sounds like an all-girls school. Nonetheless, he enjoyed the challenge of finishing an Electrical Engineering and a Computer Science double major there in May of 1987. He even completed an applied mathematics minor but fell two classes short of a literature minor.

He entered Georgia Tech's graduate program in the Fall of 1987. Being judged unworthy of department support, he found funding through the Georgia Tech Research Institute (GTRI). At GTRI, he developed knowledge bases for an expert system and wrote pattern recognition software, all in the name of optimizing the processing of poultry (chickens, that is). In December of 1988, he screeched across the stage in high-top tennis shoes to receive his M.S.E.E. degree. Upon the sudden departure of his co-advisor Dr. Schwartz, his financial support dried up in June of 1989. For the two quarters that followed, he wrote Fortran programs to help GTRI scientists analyze indoor air quality. After working for a quarter in medical expert systems, Dr. James McClellan gave him an offer he could not refuse— financial security until he graduated.

This thesis marks the end of his long trek through graduate school. His current research interests include developing computing environments for designing signal processing algorithms and exploring signal processing theory. He is a member of the Signal Processing, Circuits and Systems, and Computer Societies of the IEEE.

A Knowledge-Based Environment for the Design and Analysis of Multidimensional Multirate Signal Processing Algorithms

Brian Lawrence Evans

167 pages
Directed by Dr. James H. McClellan

This thesis discusses the design and analysis by computer of algorithms composed of linear periodically time-varying (LPTV) multidimensional systems. Analysis of linear systems is based on linear transforms (e.g. z and Laplace transforms). Algorithm design rewrites component systems to reduce the implementation cost.

To support algorithm design for multidimensional systems, the thesis derives the rules for rewriting the interconnections of discrete-time LPTV multidimensional systems, a.k.a. multidimensional multirate systems, as well as develops metrics to measure implementation costs. We encode the rewrite rules, number theoretic algorithms underlying the rules, and cost metrics in a set of multidimensional signal processing packages (MDSPPs) for the computer algebra program *Mathematica*.

For algorithm analysis, the MDSPPs implement the multidimensional multisided forms of the commonly used linear transforms, and the transforms can justify their answers with natural language. Using the transforms, the MDSPPs can deduce ranges on free parameters to guarantee stability and generate input-output relationships. Engineers can use the MDSPPs to visualize signals in transform domains.

The MDSPPs represent signals as functions and systems as operators. In such an algebraic framework, the many interconnections in complex systems cannot be captured. The MDSPPs can, however, convert an algorithm for layout in the Ptolemy block diagram environment, which fits the MDSPPs into a rapid prototyping process.