The Ultimate Algorithmic Trading System Toolbox

Using Today’s Technology To Help You Become A Better Trader

George Pruitt
THE ULTIMATE
ALGORITHMIC
TRADING SYSTEM
TOOLBOX
The Wiley Trading series features books by traders who have survived the market’s ever changing temperament and have prospered—some by reinventing systems, others by getting back to basics. Whether a novice trader, professional, or somewhere in between, these books will provide the advice and strategies needed to prosper today and well into the future. For more on this series, visit our website at www.WileyTrading.com.

Founded in 1807, John Wiley & Sons is the oldest independent publishing company in the United States. With offices in North America, Europe, Australia, and Asia, Wiley is globally committed to developing and marketing print and electronic products and services for our customers’ professional and personal knowledge and understanding.
THE ULTIMATE ALGORITHMIC TRADING SYSTEM TOOLBOX

+ Website

Using Today’s Technology to Help You Become a Better Trader

George Pruitt

WILEY
| CONTENTS |
|-------------------|------------------|
| About the author  | ix               |
| Introduction to the Ultimate Algorithmic Trading Systems Toolbox | xiii |

### CHAPTER 1 - Introduction to Trading: Algorithm Development

- What Is an Algorithm? 1
- How to Get My Trading Idea into Pseudocode 12
- Summary 23

### CHAPTER 2 - Stochastics and Averages and RSI! Oh, My!

- Oscillators 26
- Price-Based Indicators 58
- Summary 75

### CHAPTER 3 - Complete Trading Algorithms

- Trend-Trading Battle Royale 86
- Portfolio Composition 100
- Multi-Algorithm Strategy (MAS) 108
- Summary 112

### CHAPTER 4 - Introduction to AmiBroker’s AFL

- Quick Start 113
- Price Bar Interface 118
## CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFL Array Programming</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>Syntax</td>
<td>129</td>
<td></td>
</tr>
<tr>
<td>AFL Wizard</td>
<td>133</td>
<td></td>
</tr>
<tr>
<td>AmiBroker Loop Programming</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>Summary</td>
<td>140</td>
<td></td>
</tr>
</tbody>
</table>

### CHAPTER 5  Using Microsoft Excel to Backtest Your Algorithm  145

- VBA Functions and Subroutines  147
- Data  148
- Software Structure  149
- Programming Environment  154
- Summary  163

### CHAPTER 6  Using Python to Backtest Your Algorithm  167

- Why Python?  167
- Python Installation  169
- PSB Installation  169
- PSB Structure  171
- Getting Down to Business  193
- Summary  202

### CHAPTER 7  An Introduction to EasyLanguage  203

- TradeStation IDE  204
- Syntax  209
- Samples of EasyLanguage  221
- Summary  224

### CHAPTER 8  Genetic Optimization, Walk Forward, and Monte Carlo Start Trade Analysis  227

- Utilizing TradeStation and AmiBroker  227
- Computers, Evolution, and Problem Solving  230
- Population  231
- Initial Population Setup Using VBA Excel  232
- Testing Fitness of Chromosomes Using VBA Excel  232
- Selection  233
<table>
<thead>
<tr>
<th>APPENDIX D</th>
<th>TradeStation and EasyLanguage</th>
<th>323</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Importing ELD file from Book Website</td>
<td>323</td>
</tr>
<tr>
<td></td>
<td>Keywords and Functions</td>
<td>324</td>
</tr>
<tr>
<td></td>
<td>Sample Algorithm Codes</td>
<td>325</td>
</tr>
</tbody>
</table>

APPENDIX E

<table>
<thead>
<tr>
<th>About the Companion Website</th>
<th>337</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>339</td>
</tr>
</tbody>
</table>
It was March of 1989 as I drove my beat-up Dodge up Hillside Rd. in Hendersonville, NC. In an attempt to pay for my last semesters of college I was answering a classified ad that was looking to hire a computer programmer. As I drove up the thin drive I passed several houses and then through a gate attached to two large stone pillars. I stopped the car and looked down at the ad again to make sure I was at the right place. I proceeded down the country lane and the view opened up into a large meadow. At the end of the lane was a circular drive and large farm house. As I circled and went back down the road I thought to myself I must have the wrong address or directions. So I followed the small road back down the main highway and then to a small convenient store. Once there I asked myself again what type of business was this Futures Truth and if I should call and get directions or just simply forget about it. Curiosity and the need for money were too much so I used the store’s pay phone and called the number once again.

“Hello—Futures Truth, may I help you?” a lady’s voice answered.

“Yes, this is George Pruitt and I made an appointment for an interview but I can’t seem to find your office.”

“Do you drive a red Dodge?” she asked.

“Yes I do. How did you know?”

“We saw you drive right by the office. When you come through the two stone pillars turn immediately to the left. Don’t go all the way down the drive—that’s the owner’s house.”
So I follow the directions and find myself in front of a small house. I knock on the door and John Fisher opens and invites me in. We go through the normal Q and A for a job interview and he finally asks if I knew FORTRAN. My first college programming class was FORTRAN so I confidently answered, “Sure!”

He then asked me if I knew anything about the Futures market. I vaguely remembered the term from one of my economics classes and of course from the Eddie Murphy movie and answer him with the question, “You mean like Trading Places with Eddie Murphy?”

John Fisher said “Sort of like that—yes.”

He went on to explain how Futures Truth tried to determine market direction in the most widely traded futures contracts by using trading systems. The trading systems were programmed in FORTRAN and they needed help with the programming. In addition to trading they also published a newsletter in which they tracked publicly offered trading systems.

I asked, “Do people really buy these programs?”

John Fisher said yes and by that time an older gentlemen walked into the office and stated that he had spent thousands of dollars on these programs and was ultimately ripped off. John Hill stated this was the main reason he started Futures Truth. He wanted to bring truth to the trading system industry. Both Johns told me that most traders couldn’t afford to validate the trading systems because of the cost of the computer equipment, data, and software. John Fisher pointed to the computer he was working on and asked, “How much do you think this Macintosh II cost?”

I answered him, “I am not that familiar with Macs but I know they aren’t cheap.”

My mouth fell open when he said “$4,000 and we have three of them.” Remember this was way back in 1989 when computers were not cheap.

I was thinking to myself that they got ripped off because they could have got a much cheaper and better computer with the IBM PS/2. And what was up with using FORTRAN? Did they not know “C” was the new programming language of the 1990s? John Fisher chose the Apple Macintosh because of its easy-to-use graphical user interface (GUI) and FORTRAN because many traders and hobbyist programmers had knowledge of this language.

John Fisher also said that he and John Hill had developed what they considered the best testing platform, “Excalibur.” This platform could load decades of daily and intraday data and test any trading idea that could be defined in an algorithmic form. He also said the only thing that was missing was a charting application and that was where they also needed help.

I explained that I would be wrapping up my degree after summer and both Johns agreed that I could work part time in the evening until I graduated and then we could go from there.
Well that was 27 years ago and I did work part time until I graduated with a degree in computer science from the University of North Carolina at Asheville. The “Excalibur Chart” project turned into my senior project, which blew my professors away. Over the years I have worked with many trading firms in the development of trading algorithms and testing platforms. I have seen it all and have had the great pleasure to be educated by some of the greatest minds in the industry, including John Fisher, John Hill Sr. and John Hill Jr. Even with this experience and education the ultimate trading system still eludes me. As John Hill has stated many times, “A speculator who dies rich, dies before his time!” This may be true, but I have seen traders make millions, lose millions, and make millions again. The one thing they always do when they fail is get right back up, dust themselves off, and start searching for the next great trading algorithm.
If you want to learn more about High Frequency Trading utilizing special order placement/replacement algorithms such as Predatory trading, Pinging, Point of Presence, or Liquidity Rebates then this book is not for you. However, if you want to learn about trading algorithms that help make a trading decision, trade size, money management and the software used to create these algorithms then you’re in the right place.

This book is designed to teach trading algorithm development, testing and optimization. Another goal is to expose the reader to multiple testing platforms and programming languages. Don’t worry if you don’t have a background in programming this book will provide enough instruction to get you started in developing your own trading systems. Source code and instructions will be provided for TradeStation’s Easy Language, AmiBroker’s AFL, and my own Python and Excel testing engines. I chose these platforms because they give a nice overview of different scripting languages and trading platforms. Users of different testing/trading platforms may criticize my decision to use just these platforms, but the EasyLanguage source code that will be provided can easily be ported into Multi-Charts and AmiBroker’s unique and powerful platform provides a complete trading solution. My Python and Excel software, including all source code, are included on the associated website as well as the EasyLanguage and AFL source code for the other platforms. I didn’t include the use of Python’s scientific libraries, NumPy or SciPy because I wanted to keep things as simple as possible. Also I used the bare bones IDLE (Python’s own simple Integrated Development Environment) to cut down on the learning curve – I wanted to get to the bare essentials of Python without muddying the water with a sophisticated IDE. Many successful Quants utilize R
(a GNU project for statistical computing), but again to keep things simple I stuck with the easy to learn Python. The majority, if not all algorithms were tested utilizing commodity and futures data only. All the testing platforms in the book can be used to test stocks and ETFs, and all the included trading algorithms can be applied to these assets as well. Stock and ETF data is very simple to acquire. Getting commodity and futures data in an easily usable format is a little more difficult. Deep histories for commodity and futures can be acquired for as little as $100 from Pinnacle Data. I have used CSI data since the late 1980’s and it is the data I used for a good portion of the testing carried out in the book. I would definitely take a look at Pinnacle and CSI data especially if you wanted your database updated daily. If you are not familiar with Quandl, then you might want to take the time to do so. Quandl is a search engine for numerical data. I was pleasantly surprised to find a free continuous futures database (Wiki Continuous Futures) on Quandl. Keep in mind this data is free and is no way as good as premium data such as CSI and Pinnalce – its missing multiple days and data points and the continuous data is simply created by concatenating data from individual contracts. The gaps between contracts is included which cannot be utilized on any testing platform. In real life, a futures position is “rolled-over” from one contract to another by liquidating the front month position and initiating the same position in next contract. This “rollover” trade eliminates the gap. I have written a Python application that takes the Wiki Futures data and creates a back adjusted continuous contract that can be imported into the Python and Excel System Back Tester software. Since I needed the data to do testing, I have also included a 10 plus year ASCII back adjusted futures database for 30 plus markets on the companion website. Directions on how to use the software and download futures data from Quandl is included along with the software.

The one thing I really wanted to include in this book was the “Holy Grail” of algorithmic trading systems. I have analyzed many algorithms that claimed to be the “Grail”, but after rigorous testing they failed to even break even. So go ahead and check this off your list. Even though the “Holy Grail” will remain hidden you will find the following:

- twenty-seven years of experience working with non-programmers in the development of their own trading algorithms
- the tools or building blocks that are used most often in the development cycle
- the core trading models that make up the majority of publicly offered trading systems
- the most important and simplest programming techniques to transform a non-quant into a not so non-quant
simple examples and explanations of complex trading ideas such as Walk Forward
and Genetic Optimization and Monte Carlo simulation

a complete tool box to help algorithm development from idea to a finished
tradable solution

The majority of successful trading algorithms utilize quantitative analysis (QA). QA is simply the application of mathematical formulae to a financial time series. This book will solely focus on this type of analysis in the design of trading algorithms. Fundamental analysis, which is used in many trading plans, will be used too but it will be reduced and simplified into a pure and easily digestible data format. Fundamental data is huge and diverse and in many cases market movement reacts to it in an unpredictable manner. A good example that I have dealt with for many years is the monthly unemployment report. At the time of the writing of this book unemployment has been on a downward trend, which is usually a bullish indicator for the stock market. However, with interest rates at the time being close to 0% the market could react opposite due to the fear of the Federal Reserve doing away with Quantitative Easing and raising rates. This type of fundamental analysis requires many different inputs and trying to reduce it down to something testable is nearly impossible.

Quantitative analysis focuses on just the data included in a chart. Price action and price translations are easily definable and therefore can be tested. The ability to test and evaluate a trading algorithm is a tremendous tool as it shows how a model can accurately map a market’s behavior. If you can interpret a market’s behavior you then can take advantage of its inefficiencies. If an algorithm has been capable of exploiting a market’s inefficiencies on a historic basis, then there is possibility it will do so in the future. This hope of future performance is the only leg an algorithmic trader has to stand upon. We all know historic performance is not necessarily an indicator of future results, but what else do we have? An algorithmic trader that quickly defines and tests his system and immediately takes a leap of faith because the historic performance looks great is doomed. Doesn’t this contradict what I just said about historical performance being a system trader’s only gauge of quality? A good trading algorithm not only demonstrates profitability but also robustness. Robustness is an expression of how well a trading system performs on diverse markets and diverse market conditions. An algorithm can be improved to a point where the trader can feel somewhat confident putting on those first few trades as well as continuing to put trades on after a losing streak. Improving an algorithm is not simply tweaking it until the historic results look utterly fantastic (aka curve fitting); it is taking the time to learn and work with tools that are designed to make a
trading algorithm fail. That’s the ultimate objective - making your trading algorithm fail before any money is put on the line. Remember the absence of failure is success and if your algorithm survives the brutal gauntlet of in depth analysis, then you know you might, just might have a winner.

This book starts out simple in Chapter 1 with the definition and examples of algorithms. The chapter is a little long winded but I know that the inability to put a trading idea onto paper then into pseudocode and finally actual computer code is the biggest stumbling block for traders who want to test their own trading ideas. All trading algorithms that are reducible to a set of instructions can be properly programmed using one of two different modelling methods or paradigms. These two paradigms, Finite State Machine and Flow Chart, are fully discussed and utilized to translate written descriptions first into diagrams and then into actual pseudocode. The diagrammatic approach as well as the simple pseudocode language used to formulate trading algorithms is introduced in this chapter. It doesn’t matter how sophisticated your testing software is if you can’t define a testable algorithm and this chapter shows you how to do so.

Chapter 2 may be a refresher for those who are familiar with the basic building blocks of trading algorithms, indicators, however the chapter not only explains the logic behind the indicators but shows how they can be incorporated into complete entry and exit techniques. Diagrams and pseudocode are carried on through this chapter to aid in the understanding of each indicator, its purpose and its place in a trading algorithm. In addition, the first look at indicator based trading algorithm performance is presented as well.

Chapter 3 introduces complete trading algorithms and their associated historical performance. Most, if not all, testing was performed on historical commodity/futures data. This data gave rise to the concept of systematic trading more fifty years ago. Now this doesn’t mean the ideas aren’t transferable to the stock market. In most cases they are. However, I stuck with commodity data because that is where my expertise lies. The complete pseudocode and actual computer code of these algorithms are revealed as well. The key metrics for determining algorithm robustness are explained and utilized in the evaluation of the algorithms results.

Chapter 4 starts the section that highlights different testing/trading software platforms that can either be purchased or leased. AmiBroker is introduced in this chapter and the most important components of a trading platform are highlighted: integrated development environment and its associated scripting/programming language, individual market and portfolio testing, and algorithm performance metrics. These components are then highlighted again in Chapter 5 with VBA for Excel, Chapter 6 with Python and then finally Chapter 7 with TradeStation.

Chapter 8 delves into the concepts of Genetic and Walk Forward Optimization, Walk Forward Analysis and Monte Carlo simulation. A genetic optimizer is built using VBA and used to help explain the ideas of synthesizing computers with
The core concepts of Genetic Algorithms, fitness, selection, reproduction and mutation are fully explained and illustrated utilizing Excel. Artificial Intelligence is here to stay in the study of trading algorithms and this chapter tries to pull back the veil of mystery and show these tools should be used, and in some cases, must be used to develop that all elusive robustness. Along these lines, Machine Learning has become a very highly discussed and somewhat controversial topic in today’s trading. Also “Big Data” analysis has found its way to the front as well. These topics are highly advanced and I felt beyond the scope of this book. I can state I have worked with the algorithms that were derived with machine only input and they have stood the test of time.

A trading algorithm must work over a diverse portfolio of markets before it can be sufficiently considered useful and robust. Chapter 9 utilizes the portfolio level testing capabilities of TradeStation and AmiBroker to demonstrate different money and portfolio management techniques. The Fixed Fractional approach, by far the most popular, will be highlighted.

The complete source code for the Python System Back Tester is included on the website. Python is the new language of many a quant and the source shows how the language can be used to develop a simple, yet powerful, back tester. Important language concepts and syntax are used to open ASCII files, and import the data into a LIST data structure, create classes and modules, and loop through the entire database while applying a trading algorithm. All the parts of building a testing platform are revealed in the source code including Monte Carlo and Start Trade Drawdown simulation.

Most traders have Microsoft Excel on their computers and the complete source for a more simplified version of the Python back tester using VBA is included on the website as well.

This book is a toolbox and a guide and touches upon many different facets of algorithmic trading. As with any toolbox it will take time and effort to apply the tools found within to replicate the trader’s ideas in a form that not only can be tested and evaluated but fully implemented.
What Is an Algorithm?

An Algorithm is an effective procedure, a way of getting something done in a finite number of discrete steps.

David Berlinski

Berlinski’s definition is exactly right on the money. The word algorithm sounds mysterious as well as intellectual but it’s really a fancy name for a recipe. It explains precisely the stuff and steps necessary to accomplish a task. Even though you can perceive an algorithm to be a simple recipe, it must, like all things dealing with computers, follow specific criteria:

1. Input: There are zero or more quantities that are externally supplied.
2. Output: At least one quantity is produced.
3. Definiteness: Each instruction must be clear and unambiguous.
4. Finiteness: If we trace out the instructions of an algorithm, then for all cases the algorithm will terminate after a finite number of steps.
5. **Effectiveness**: Every instruction must be sufficiently basic that it can in principle be carried out by a person using only pencil and paper. It is not enough that each operation be definite as in (3), but it must also be feasible. [Fundamentals of Data Structures: Ellis Horowitz and Sartaj Sahni 1976; Computer Science Press]

These criteria are very precise because they can be universally applied to any type of problem. Don’t be turned off thinking this is going to be another computer science text, because even though the criteria of an algorithm seem to be very formal, an algorithm really is straightforward and quite eloquent. It is basically a guide that one must follow to convert a problem into something a computer can solve. Anybody can design an algorithm following these criteria with pencil and paper. The only prerequisite is that you must think like a Vulcan from Star Trek. In other words, think in logical terms by breaking ideas down into rudimentary building blocks. This is the first step—translation of idea into an algorithm. It takes practice to do this, and this is in part why programming can be difficult.

Another thing that makes programming difficult is understanding a computer language’s syntax. Most people who have been exposed to a programming or scripting language at one time or another in their lives have probably exclaimed something like, “I forgot one stupid semicolon and the entire program crashed! Isn’t the computer smart enough to know that? Argh! I will never be a computer programmer!” The question that is proffered in this temper tantrum is the question of the computer’s intelligence. Computers are not smart—they only do what we tell them. It doesn’t matter if you spend $500 or $5,000 on the hardware. They do things very quickly and accurately, but their intelligence is a reflection of their programmer’s ability to translate idea into algorithmic form and then into proper syntax.

Algorithmic (algo) traders don’t necessarily need to be programmers, but they must understand what a computer needs to know to carry out a trading signal, position sizing, and money management. If you can create an algorithm, then you are more than 50 percent there. I say more than 50 percent because most traders will utilize trading software and its associated programming or scripting language. Learning the syntax of a scripting language or a small subset of a programming dialect is much easier than learning an entire programming language like C# or C++. An algo trader only needs to be concerned with the necessary tools to carry out a trading system. The developers of EasyLanguage, AmiBroker, or TradersStudio’s main objective was to provide only the necessary tools to put a trading idea into action. They accomplished this by creating a vast library of trading functions, easy access to these functions, and a simplified programming syntax. Now if you want to develop your own testing platform and want to use a full-blown programming language to do so, then you will need to know the language inside-out. If you are interested in doing this, Chapters 5 and 6 will give you a head start. In these chapters, I show how I developed testing platforms in Python and Microsoft VBA from scratch.
However, at this introductory stage, let’s take a look at a very simple trading algorithm and the possible exchange between a computer and trader. Pretend a trader wants to develop and test a simple moving-average crossover system and wants to use software specifically designed for system testing. Let’s call this first trader AlgoTrader1, and since he has used this particular testing platform he knows it understands a trading vernacular and provides access to the common indicator functions and data. Box 1.1 shows a possible exchange between trader and computer.

Box 1.1 Algo Testing Software

<table>
<thead>
<tr>
<th>AlgoTrader1</th>
<th>Computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlgoTester ON</td>
<td>AlgoTester ready</td>
</tr>
<tr>
<td>AlgoTrader1 - load crude oil futures data</td>
<td>Computer - data loaded</td>
</tr>
<tr>
<td>AlgoTrader1 - buy whenever close is above moving average</td>
<td>Computer - &quot;moving average&quot; function requires three inputs</td>
</tr>
<tr>
<td>AlgoTrader1 - help with moving average function</td>
<td>Computer - function calculates simple, weighted, exponential average</td>
</tr>
<tr>
<td>AlgoTrader1 - function syntax moving average (type, price, length)</td>
<td>Computer - command completed</td>
</tr>
<tr>
<td>AlgoTrader1 - buy whenever close is above moving average (simple, close, 21)</td>
<td>Computer - command completed</td>
</tr>
<tr>
<td>AlgoTrader1 - short whenever close is below moving average (simple, close, 21)</td>
<td>Computer - command completed</td>
</tr>
<tr>
<td>AlgoTrader1 - save algorithm as MovAvgCross</td>
<td>Computer - command completed</td>
</tr>
<tr>
<td>AlgoTrader1 - run MovAvgCross algorithm</td>
<td>Computer - run completed and results are: $12,040 profit, $8,500 draw down, $1,200 avg. win</td>
</tr>
<tr>
<td>AlgoTrader1 - load euro currency data</td>
<td>Computer - command completed</td>
</tr>
<tr>
<td>AlgoTrader2 - run MovAvgCross algorithm</td>
<td>Computer - run completed and results are: -$32,090 profit, $40,000 draw down, $400 avg. win</td>
</tr>
<tr>
<td>AlgoTrader1 - edit MovAvgCross algorithm</td>
<td>Computer - command completed</td>
</tr>
<tr>
<td>AlgoTrader2 - edit moving average function</td>
<td>Computer - command completed</td>
</tr>
<tr>
<td>AlgoTrader2 - change length input to 30</td>
<td>Computer - command completed</td>
</tr>
<tr>
<td>AlgoTrader2 - run MovAvgCross algorithm</td>
<td>Computer - run completed and blah blah blah</td>
</tr>
</tbody>
</table>
As you can see, the computer had to be somewhat spoon-fed the instructions. The software recognized many keywords such as: load, buy, short, run, edit, change, and save. It also recognized the moving average function and was able to provide information on how to properly use it. The trading algorithm is now stored in the computer’s library and will be accessible in the future.

This simple exchange between computer and AlgoTrader1 doesn’t reveal all the computations or processes going on behind the scene. Loading and understanding the data, applying the algorithm properly, keeping track of the trades, and, finally, calculating all of the performance metrics did not involve any interaction with the trader. All this programming was done ahead of time and was hidden from the trader and this allows an algo trader to be creative without being bogged down in all the minutiae of a testing platform.

Even though the computer can do all these things seamlessly it still needed to be told exactly what to do. This scenario is similar to a parent instructing a child on how to do his first long-division problem. A child attempting long division probably knows how to add, subtract, multiply, and divide. However, even with these “built-in” tools, a child needs a list of exact instructions to complete a problem. An extended vocabulary or a large library of built-in functions saves a lot of time, but it doesn’t necessarily make the computer any smarter. This is an example of knowledge versus intelligence—all the knowledge in the world will not necessarily help solve a complex problem. To make a long story short, think like a computer or child when developing and describing a trading algorithm. Box 1.2 shows an algorithmic representation of the long-division process to illustrate how even a simple process can seem complicated when it is broken down into steps.

### Box 1.2 Procedure for Long Division

Suppose you are dividing two large numbers in the problem \( n \div m \). In this example, the dividend is \( n \) and the divisor is \( m \).

If the divisor is not a whole number, simplify the problem by moving the decimal of the divisor until it is to the right of the last digit. Then, move the decimal of the dividend the same number of places. If you run out of digits in the dividend, add zeroes as placeholders.

When doing long division, the numbers above and below the tableau should be vertically aligned.

Now you are ready to divide. Look at the first digit of the dividend. If the divisor can go into that number at least once, write the total number of times it fits
completely above the tableau. If the divisor is too big, move to the next digit of the dividend, so you are looking at a two-digit number. Do this until the divisor will go into the dividend at least once. Write the number of times the divisor can go into the dividend above the tableau. This is the first number of your quotient.

Multiply the divisor by the first number of the quotient and write the product under the dividend, lining the digits up appropriately. Subtract the product from the dividend. Then, bring the next digit from the quotient down to the right of the difference. Determine how many times the divisor can go into that number, and write it above the tableau as the next number of the quotient.

Repeat this process until there are no fully divisible numbers left. The number remaining at the bottom of the subtraction under the tableau is the remainder. To finish the problem, bring the remainder, \( r \), to the top of the tableau and create a fraction, \( r/m \).

A few years ago a client came to me with the following trading system description and hired me to program it. Before reading the description, see if you can see any problems the programmer (me) or a computer might encounter before the directives can be properly carried out.

Buy when the market closes above the 200-day moving average and then starts to trend downward and the RSI bottoms out below 20 and starts moving up. The sell short side is just the opposite.

Did you see the problems with this description? Try instructing a computer to follow these directions. It doesn’t matter if a computer has a vast library of trading functions; it still would not understand these instructions. The good news was, the trader did define two conditions precisely: close greater than 200-day moving average and relative strength index (RSI) below 20. The rest of the instructions were open to interpretation. What does trending downward or bottoming out mean? Humans can interpret this, but the computer has no idea what you are talking about. I was finally able, after a couple of phone calls, to discern enough information from the client to create some pseudocode. Pseudocode is an informal high-level description of the operating principle of a computer program or algorithm. Think of it as the bridge between a native language description and quasi-syntactically correct code that a computer can understand. Translating an idea into pseudocode is like converting a
nebulous idea into something with structure. Here is the pseudocode of the client’s initial trading idea:

**Algorithm Pseudocode**

```plaintext
if close > 200 day moving average and 
close < close[1] and close [1] < close [2] and 
close[2] < close[3] and yesterday’s 14 day RSI < 20 and 
yesterday’s 14 day RSI < today’s 14 day RSI then BUY
```

If this looks like Latin (and you don’t know Latin), don’t worry about it. The [1] in the code just represents the number of bars back. So close [2] represents the close price prior to yesterday. If you are not familiar with RSI, you will be after Chapter 2. By the time you are through with this book you will be able to translate this into English, Python, EasyLanguage, AmiBroker, or Excel VBA. Here it is in English.

If today’s close is greater than the 200-day moving average of closing prices and today’s close is less than yesterday’s close and yesterday’s close is less than the prior day’s close and the prior day’s close is less than the day before that and the 14-day RSI of yesterday is less than 20 and the 14-day RSI of yesterday is less than the 14-day RSI of today, then buy at the market.

Notice how the words *downward* and *bottoming out* were removed and replaced with exact descriptions:

- **downward**: today’s close is less than yesterday’s close and yesterday’s close is less than the prior day’s close and the prior day’s close is less than the day before. The market has closed down for three straight days.

- **bottoming out**: the 14-day RSI of yesterday is less than 20 and the 14-day RSI of yesterday is less than the 14-day RSI of today. The RSI value ticked below 20 and then ticked up.

Also notice how the new description of the trading system is much longer than the original. This is a normal occurrence of the evolution of idea into an exact trading algorithm.

And now here it is in the Python programming language:

**Actual Python Code**

```python
if myClose[D0] < sAverage(myClose,200,D0,1) and 
myClose[D0] < myClose[D1] and myClose[D2] < myClose[D3] and 
myClose[D1] < myClose[D2] and rsiVal[D1] < 20 and rsiVal[D1] 
< rsiVal[D0]: 
    buyEntryName = 'rsiBuy'
    entryPrice = myOpen
```
Don’t get bogged down trying to understand exactly what is going on; just notice the similarity between pseudo and actual code. Now this is something the computer can sink its teeth into. Unfortunately, reducing a trading idea down to pseudocode is as difficult as programming it into a testing platform. The transformation from a trader to an algo trader is very difficult and in some cases cannot be accomplished. I have worked with many clients who simply could not reduce what they saw on a chart into concise step-by-step instructions. In other cases the reduction of a trading idea removes enough of the trader’s nuances that it turned something that seemed plausible into something that wasn’t.

It goes without saying that if you don’t have an algorithm, then all the software in the world will not make you a systematic trader. Either you have to design your own or you have to purchase somebody else’s. Buying another person’s technology is not a bad way to go, but unless the algorithm is fully disclosed you will not learn anything. However, you will be systematic trader. I have spent 27 years evaluating trading systems and have come across good and bad and really bad technology. I can say without a doubt that one single type of algorithm does not stand head and shoulders above all the others. I can also say without a doubt there isn’t a correlation between the price of a trading system and its future profitability. The description in Box 1.3 is very similar to a system that sold for over $10,000 in the 1990s.

Box 1.3 Trading Algorithm Similar to One That Sold for $10K in the 1990s Description

**Entry Logic:**

If the 13-day moving average of closes > the 65-day moving average of closes and the 13-day moving average is rising and the 65-day moving average is rising then buy the next day’s open

If the 13-day moving average of closes < the 65-day moving average of closes and the 13-day moving average is falling and the 65-day moving average is falling then sell the next day’s open

**Exit Logic:**

If in a long position then

set initial stop at the lowest low of the past 13 days

If in a short position then

set initial stop at the highest high of the past 13 days

Once profit exceeds or matches $700 pull stops to break even
If in a long position use the greater of:
- Breakeven stop—if applicable
- Initial stop
- Lowest low of a moving window of the past 23 days

If in a short position use the lesser of:
- Breakeven stop—if applicable
- Initial stop
- Highest high of the moving window of the past 23 days

That is the entire system, and it did in fact sell for more than $10K. This boils down to a simple moving-average crossover system trading in the direction of the shorter- and longer-term trend. The description also includes a complete set of trade management rules: protective, breakeven, and trailing stop. This is a complete trading system description, but as thorough as it seems there are a couple of problems. The first is easy to fix because it involves syntax but the second involves logic. There are two words that describe market direction that cannot be interpreted by a computer. Do you see them? The two words in question are: rising and falling. Again, descriptive words like these have to be precisely defined. This initial problem is easy to fix—just inform the computer the exact meaning of rising and falling. Second, it has a weakness from a logical standpoint. The algorithm uses $700 as the profit level before the stop can be moved to break even. Seven hundred dollars in the 1990s is quite a bit different than it is today. The robustness of this logic could be dramatically improved by using a market-derived parameter. One could use a volatility measure like the average range of the past N-days. If the market exhibits a high level of volatility, then the profit objective is larger and the breakeven stop will take longer to get activated. You may ask, “Why is this beneficial?” Market noise is considered the same as volatility, and the more noise, the higher likelihood of wide market swings. If trading in this environment, you want to make sure you adjust your entries and exits so you stay outside the noise bands.

This algorithm was very successful in the 1980s and through a good portion the 1990s. However, its success has been hit-and-miss since. Is this algorithm worth $10K? If you were sitting back in 1993 and looked at the historical equity curve, you might say yes. With a testing platform, we can walk forward the algorithm, and apply it to the most recent data and see how it would have performed and then answer the question. This test was done and the answer came out to be an emphatic no!

Had you bought this system and stuck with it through the steep drawdowns that have occurred since the 1990s, you would have eventually made back your investment (although not many people would have stuck with it). And you would have learned the secret behind the system. Once the secret was revealed and your
checking account was down the $10K, you might have been a little disappointed knowing that basic tenets of the system had been around for many years and freely disseminated in books and trade journals of that era. The system may not be all that great, but the structure of the algorithm is very clean and accomplishes the tasks necessary for a complete trading system.

The most time-consuming aspect when developing a complete trading idea is coming up with the trade entry. This seems somewhat like backward thinking because it’s the exit that determines the success of the entry. Nonetheless, the lion’s share of focus has always been on the entry. This system provides a very succinct trade entry algorithm. If you want to develop your own trading algorithm, then you must also provide the computer with logic just as precise and easy to interpret. Getting from the nebula of a trading idea to this point is not easy, but it is absolutely necessary. On past projects, I have provided the template shown in Box 1.4 to clients to help them write their own entry rules in a more easily understood form. You can download this form and a similar exit template at this book’s companion website: www.wiley.com/go/ultimatealgotoolbox.

**Box 1.4 Simple Template for Entry Rules**

**Long / Short Entries**

Calculations and/or Indicators Involved (specify lookback period). Don’t use ambiguously descriptive words like *rising*, *falling*, *flattening*, *topping* or *bottoming out*.

___________________________________________________

___________________________________________________

___________________________________________________

Buy Condition—What must happen for a long signal to be issued? List steps in chronological order. And remember, don’t use ambiguously descriptive words like *rising*, *falling*, *flattening*, *topping*, or *bottoming out*.

___________________________________________________

___________________________________________________

___________________________________________________

Sell Condition—What must happen for a short signal to be issued? List steps in chronological order.

___________________________________________________

___________________________________________________

___________________________________________________

Here is one of the templates filled out by a one-time client:

Calculations and/or Indicators Involved (specify lookback period)

Bollinger Band with a 50-day lookback
Buy Condition—What must happen for a long signal to be issued? List steps in chronological order.

1. Close of yesterday is above 50-day upper Bollinger Band
2. Today’s close < yesterday’s close
3. Buy next morning’s open

Sell Condition—What must happen for a short signal to be issued? List steps in chronological order.

1. Close of yesterday is below 50-day lower Bollinger Band
2. Today’s close > yesterday’s close
3. Sell next morning’s open

The simple Bollinger Band system shown in Box 1.4 is seeking to buy after the upper Bollinger Band penetration is followed by a down close. The conditions must occur within one daily bar of each other. In other words, things must happen consecutively: the close of yesterday is > upper Bollinger Band and close of today < yesterday’s close. The sell side of things uses just the opposite logic. The template for exiting a trade is very similar to that of entering. Box 1.5 contains a simple template for exit rules and shows what the client from Box 1.4 had completed for his strategy.

**Box 1.5 Simple Template for Exit Rules**

**Exits**

Calculations and/or Indicators Involved (specify lookback period). Don’t use ambiguously descriptive words like rising, falling, flattening, topping, or bottoming out.

___________________________________________________
___________________________________________________
___________________________________________________

Long Liquidation Condition—What must happen to get out of a long position? List steps in chronological order.

___________________________________________________
___________________________________________________
___________________________________________________