FABIÁN ROMERO

PURELY FUNCTIONAL, FORMALLY VERIFIED FINANCIAL LIBRARY.

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PUBLISHED BY POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

TUFTE-LATEX.GOOGLECODE.COM

Contents

Summary

The goal of this project is to describe an implementation of a functional library for finance calculation. Applying the current trends and ideas of research in functional programming, program verification and communication verification in distributed computing. Aiming not only to create a coherent library based on first principles. But also to preserve speed and accuracy similar to the best solutions currently used in this demanding industry. It is also completely possible that the purelly functional data structures currently known may or may not cover all the needs for such library and quite possibly be the case for current implementations on session types to not to be up to speed. Therefore this project also targets to improve the current implementations, tools, data structures and algorithms to achieve its stated goals.

Background

There are many libraries for finance. Most of them are proprietary and programmed in "close to the metal" languages such as c++. This is because speed and accuracy play an important role in finance, more so in the valuation and trading areas.

In the case of stock valuation several goals collide. The analysis is hard due to the complex, interdependent and concurrent behaviour. Also, speed happens to be just as important. High frequency trading (HFT) is the norm. Having a valuation done faster or more accurately than that of a competitor yields a vast, potentially insurmountable advantage. Even if it is just a μs faster or a $\mu £$ more accurate.

Besides speed, another desirable trait is correctness. But it has been long thought that it is unfeasible to achieve it. So, the industry has settled with lesser goals, such as unit testing, model validation and extensive debugging.

Summarizing, the status of software development in this industry is focused on speed and getting the job done. Sidelining a coherent, more comprehensive approach, based on firsts principles.

On the other hand. Hardware is getting only better. Processing power is enormous due to virtualisation, multicore, paralellisation and new "cloud" computing techniques. Software advances are just as big. More capable compilers and more sophisticated techniques allow us to use effectively such hardware. This is especially true for Functional Programming (FP). And particularly for Purely Functional Data Structures (PFDS). Also, formal proof systems have made a lot of progress. Today we can write formally proved correct software, timely and in budget.

As well as computing power increases, distributed systems are fast becoming the norm. Session types are designed to specify communication protocols and typed data exchanges. Having matured in its simple yet expressive elegance. They allow us to describe and think algebraically about concurrent systems. Guaranteeing communication safety, fidelity and type soundness.

Sadly most of the current libraries of ***PFDS*** are used in "domestic consumption". Almost exclusively by ***FP*** researchers and enthusiasts.

Introduction

Purely functional data structures

As mentioned, vast and impressive advances have been made in compiling techniques, however as [1] Okasaki remarks on his now classic book "no amount of cleverness on the part of the compiler writer is likely to mitigate — the use of inferior or inappropriate data structures".

[1] Chris Okasaki. *Purely functional data structures*. Cambridge University Press, 1999. ISBN 978-0-521-66350-2

Purely functional data structures, as defined on Okasaki's book, are immutable data sctructures, this simple fact brings many advantages, among them:

- **Thread safety** Locking is unnecesary, so concurrency is enourmously simplified
- **Sharing** Since inmutability assures no one of the structures involved will destroy or make changes, sharing substructures is possible.
- **Simplicity** Not having to keep track of state changes, allows to think more clearly about them, we can do even purelly algebraic reasoning.
- **Simplified Distribution** As Michel Raynal [2] ponted out, "an execution (of a distributed program) is a partial order on operations". However thanks to referential transperency in a pure functional program, the order of execution of side effect free functions is not important, and such ordering irrelevant.

[2] Michel Raynal. *Distributed Algorithms for Message-Passing Systems*. Springer, 2013. ISBN 978-3-642-38122-5

Apparently a hefty price has to be paid for this advantages in terms of memory and efficiency. Because ephemeral data scrutures, only have to use the exact ammount of memory the "current state" needs. Fortunately this is not allways the case, in many cases we can reconcile amortization and persistence by using lazy evaluation, which is possible even in strictly evaluated functional languages. Therefore, it is often possible to create data structures that are asymptotically as efficient as the best imperative solutions.

Structured finance

The essence of structured finance is the pooling of economic assets such as loans, bonds, mortgages, etc, and subsequent issuance of claims against them (tranches) using a prioritized capital structure. Those prioritization schemes used in structuring are created to repackage risks. As result, some tranches are safer than the average asset in the underlying pool. The resulting tranches are sold in the market as derivative securities.

The structuring, valuation and analysis of those securities, specifically Colletarized Debt Obligations (CDO) in residential mortgage backed securities (RMBS) has played a central role on the finance world, because as noted on Fabozzi's book [3] "the market of mortgage based securities has become the largest cash financial market in the world" whose value is in the order of several trillion USD.

As expected, the inherent risks of such massive market is enourmous, and, as exposed with 2007-2008 events can potentially collapse the world's economy.

It is very hard to overstate the importance of the correct assesment of value and risk of such assets.

Also, because of this importance, many new techniques, methods and theory have emerged on this field over the last decade, such methods have been crossing boundaries over other fields of finance and can be used as a model for risk quantification and valuation for many other assets.

[3] F.J. Fabozzi, A.K. Bhattacharya, and W.S. Berliner. *Mortgage-Backed Securities: Products, Structuring, and Analytical Techniques*. Frank J. Fabozzi Series. Wiley, 2010. ISBN 9781118044711

Formal verification

Software bugs have a combined cost of hundreds of millions of dollars, many human causalities, disruption of essential services, and virtually uncountable other lesser damages.

The finance industry has been hit hard and often for software problems leading to scandalous losses, featuring in blogs and news headlines. For example, few recent events are:

- "How to lose $172,222 a second for 45 minutes".
- "minute by minute: Here's how NASDAQ screwed up the facebook IPO",
- "Finance software bug causes $217m in investor losses"

A study by NIST, "The Economic Impacts of Inadequate Infrastructure for Software Testing"[4] estimates the cost of software errors in the USA only, to be about sixty billion dollars annually. It is a large figure indeed, but they conspicuously assumed that software testing is implicitly the only way to reduce software defects. This is a long held belief. To cite Brook [5] "Verifications are so much work that only a few substantial programs have ever been verified". Which of course was true at the time. But even today, most people in the sofware industry, practitioners and academics alike, keep assuming that the cost of formal program verification outweigh its benefits, some people go even further and mistakenly say it is impossible. But testing alone it is an inadequate method in the pursuit of reliability, as Dijkstra clearly stated [6] "Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence. The only effective way to raise the confidence level of a program significantly is to give a convincing proof of its correctness".

Formal verification has made great strides, today it is completely possible to create verified software within sensible effort. Moreover, in mission critical software it is only reasonable and responsible to do so. By applying formal verification techniques, it can be proved, with mathematical certainty, that the code produced behaves exactly as specified.

[4] G. Tassey. The economic impacts of inadequate infrastructure for software testing, 2002

[5] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition*. Addison-Wesley Professional, August 1995. ISBN 0201835959

[6] Edsger W. Dijkstra. The humble programmer. *Commun. ACM*, 15(10): 859–866, October 1972. ISSN 0001-0782. DOI: 10.1145/355604.361591. URL http://doi.acm.org/10.1145/355604.361591

High frecuency trading (HFT)

As nobel laureated Professor Vernon L. Smith explains [7], humans have emotional bias such as personal preferences, herd behaviour, panic, stress induced performance degradation, etc. Which hinders sound decision making. Not to mention to be outrigth uncapable to react in the millisecond time scale.

However, computers can trade reactively, accurately and unemotionally in that environment. Unsurprisingly they have took the lead in the trading floor. High-frequency trading (HFT) is the use of computer algorithms to rapidly exchange securities, by the millisecond. To let it clear, it is required that in a millisecond time to:

- Read the market
- Assess the value of a stock (and correlated stocks)
- Comprehend trends of the market.
- Evaluate the oportunity of a position to be made
- Analize the value and the volume of that position
- Validate the proposed position under the risk quantification, exit strategy and other safety restrictions.
- Send the position (ask or bid).

Beside the obvious technical challenges. This kind of trading is remarkably succesfull. Point in case, one among many HFT shops named Virtu, in 4 years of trading has had only one day in which it lost money [8], their daily income is about two million USD, with a comfortable operating marging of 55%.

To put in perspective, no single stock, stock market or portfolio in the world has had this "strike it lucky", therefore there is not a single traditional long term inverstor wich could possibly have this low risk. Also $S\&P500$ average operating margin is of 10% [9].

This is because high-frequency traders, place extremelly short term positions, aiming to capture just a fraction of a cent in profit on every trade. HFT firms do not employ significant leverage, accumulate positions or even hold portfolios overnight, so they virtually have no market assets, as a result. HFT has a risk to reward ratio thousands of times higher than traditional buy-and-hold strategies.

[7] V.L. Smith. *Rationality in Economics: Constructivist and Ecological Forms*. Cambridge University Press, 2009. ISBN 9781139466462

[8] Inc. Virtu Financial. IPO prospectus. https://www.sec.gov/Archives/edgar/data/1592386/000104746914002070/a2218589zs-1.htm, 2014

[9] Inc Yardeni Research. S&P 500 Sectors and Industries Profit Margins. http://www.yardeni.com/pub/sp500margin.pdf, 2014

Session types

Very often, theory is way ahead of the technology needed to use it. Theoretical computing had its beginnings in mathematical logic through the attempt to formalize the notion of effective computability. Well before the construction of computers. Turing machines, lambda calculus, recursive functions and many other models where proposed. Church's Lambda calculus was specially appealing to logicians and mathematicians because to its simplicity and power, consisting in only one transformation rule and a single function definition scheme. All those models happened to be equivalent. And layed the fundations for the information revolution to come.

Sometimes technology is ahead of theory, and in the case of distributed computing many issues like task-scheduling in operating systems, asynchronous computation, protocols over networks, communication delays, failures of communication or processors and many others. Where present and no sound theory was readily available to solve them.

Was then evident the need for a theory. The original goals of this new field were [10]: "To build a mathematical theory of distributed computing that would shed light on distributed computing systems just as Turing machine theory had done for sequential computers. The hope was to find an abstract distributed model that would capture the salient features of real distributed systems while suppressing distracting and unenlightening details of the physical world. The theory to be constructed was to be elegant, general, and powerful".

[10] Michael J. Fischer and Michael Merritt. Appraising two decades of distributed computing theory research. *Distributed Computing*, 16(2-3):239–247, 2003

One aproach in that quest was process calculi formally modelling concurrent systems. Process calculi provide a tool for the high-level description of interactions, communications, and synchronizations between a collection of independent agents or processes. Also provide algebraic laws that allow process descriptions to be manipulated and analyzed. One of the most succesfull among them was π-calculus, but it lacked verificability.

One refinment latter over *pi*-calculus are session types which allow plans of conversation between two concurrent processes to be treated as types. Type checking then ensures that communication between processes is safe: i.e. it obeys the protocol specified by the session type. Thus session types offer a means to establish conformance to protocols in distributed applications.

Objectives

Expected Outcomes

nothing

we think of error freeness checked via typability, of internal mobility which nicely captures the idea of private conversations, of linearity and type duality which enforce the mirroring of the channel usage into its type, and of channel transmission, at the very basis of the pi-calculus, to model service dele- gation. Session Types

A complete library can be done using only (***PFDS***). Formally proving most (all?) of their algorithms. Session types and ***PFDS*** can be successfully implemented on mission critical financial problems. Preserving the speed and accuracy of current methods. Offering the solid foundations of ***FP*** and providing some advantages as type sound concurrency. Creating a more modern, scalable, safe and coherent approach.

Using ***PFDS*** will likely provide many benefits. Among them, will simplify parallelism and concurrency. ***PFDS*** enable algebraic reasoning over data structures. Which is a useful feature given the sophistication of some calculus. Since ***PFDS*** are immutable, locking is unnecessary. To create fast software, we can't make hardware go faster. yet we can make software to achieve more with the same hardware, working less to achieve its goal. So, using a lazy language will prevent for working in something unnecessary. And since ***PFDS*** are persistent, many recalculations will be avoided.

Bibliography

Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition*. Addison-Wesley Professional, August 1995. ISBN 0201835959.

Edsger W. Dijkstra. The humble programmer. *Commun. ACM*, 15(10): 859–866, October 1972. ISSN 0001-0782. DOI: 10.1145/355604.361591. URL http://doi.acm.org/10.1145/355604.361591.

F.J. Fabozzi, A.K. Bhattacharya, and W.S. Berliner. *Mortgage-Backed Securities: Products, Structuring, and Analytical Techniques*. Frank J. Fabozzi Series. Wiley, 2010. ISBN 9781118044711.

Michael J. Fischer and Michael Merritt. Appraising two decades of distributed computing theory research. *Distributed Computing*, 16(2-3): 239–247, 2003.

Chris Okasaki. *Purely functional data structures*. Cambridge University Press, 1999. ISBN 978-0-521-66350-2.

Michel Raynal. *Distributed Algorithms for Message-Passing Systems*. Springer, 2013. ISBN 978-3-642-38122-5.

V.L. Smith. *Rationality in Economics: Constructivist and Ecological Forms*. Cambridge University Press, 2009. ISBN 9781139466462.

G. Tassey. The economic impacts of inadequate infrastructure for software testing, 2002.

Inc. Virtu Financial. IPO prospectus. https://www.sec.gov/Archives/edgar/data/1592386/000104746914002070/a2218589zs-1.htm, 2014.

Inc Yardeni Research. S&P 500 Sectors and Industries Profit Margins. http://www.yardeni.com/pub/sp500margin.pdf, 2014.