AESTHETIC INTEGRATION

Aesthetic Integration Ltd.
Level 30
122 Leadenhall Street
London, EC3V 4AB
United Kingdom

November 18th, 2015

Brent J. Fields
Secretary
Securities and Exchange Commission
100 F Street, N.E.
Washington, D.C., 20549-1090

Re: Investors' Exchange LLC Form 1 Application (Release No. 34-75925; File No. 10-222)

Dear Mr. Fields,

Although we have missed the official deadline for public comments, we would like to share our observations on IEX's application and the public comments made thus far. In particular, we would like to bring your attention to breakthrough scientific tools that will give the SEC unprecedented powers for the oversight of exchanges and dark pools. We hope you will find our feedback useful.

The task ahead of the Commission is difficult. You must analyze IEX's application to understand the possible behaviors of their system design, and determine whether or not it satisfies your regulatory requirements. If the application is successful and IEX obtains the status of an exchange, the Commission will be further tasked with the oversight of IEX's ongoing operations under its new mandate.

As recent regulatory actions by the SEC demonstrate, it currently takes the Commission considerable time and resources to analyze a trading venue for flaws in its design and implementation. In recent public cases (e.g., the UBS ATS settlement from earlier this year), the cited wrongdoings and subsequent investigations span many years. Processing applications like IEX's and ensuring complex venue algorithms operate correctly is a highly non-trivial task. But, with recent scientific breakthroughs in the field of 'formal verification,' this process can be significantly improved and automated. Other safety-critical industries like avionics and hardware manufacturing already rely upon related techniques to analyze and regulate their complex algorithms. As an industry, we can learn a lot from their experiences.

Ultimately, the concerns raised in public comments center around the possible behavior of algorithms implemented within IEX production systems. There is much speculation about their fairness and their more general effects on market microstructure. For example, questions have been raised regarding access to exchange data by the router component of the IEX brokerage, IEXS. Another set of questions concerns transparency of order types discussed within the application.

By applying modern formal verification techniques developed precisely for reasoning about complex algorithms, you can perform much deeper analyses, save taxpayers a lot of money, and significantly improve the transparency and stability of our financial markets.

The Source of Complexity

In our view, the numerous questions surrounding the IEX application (and also the regulatory challenges involved in processing such applications) stem from the following three fundamental issues:

1. The current approach for both disclosing and analyzing venue designs is inappropriate for the complexity of modern financial markets. A venue design is an incredibly complex algorithm. Given its collection of order types, transitions into volatility auctions, circuit-breakers, etc., the set of its possible behaviors (`state-space') is virtually infinite. This state-space simply cannot be exhaustively examined without powerful tools for the analysis of algorithms.

   Asking venue operators to describe their designs in English prose significantly hinders their ability to accurately describe their systems. Moreover, it significantly hinders your ability to analyze the design for compliance with regulatory directives. To reiterate, this is not necessarily about 'what' exchanges and dark pool operators should disclose, but rather 'how' they should disclose it. The format matters. English prose is not a proper format for disclosing complex algorithms that must be analyzed for regulatory requirements.

2. The Commission currently has no way to automatically connect venue designs to the actual venue implementations. You simply cannot reconcile documents such as exchange by-laws or Form ATSs with the actual post-trade data. You cannot 'execute' such a regulatory submission and check to see if post-trade data matches the logic described in the venue design. This puts a significant burden on regulators to analyze data, without a proper analysable specification of what the venue should be doing.

3. The industry lacks quantitative metrics for expressing the sufficiency of system testing. Many recent regulatory directives discuss the requirement for 'sufficient testing.' But no financial regulator has defined precisely what that means. Financial regulators are behind the times in this respect. In avionics, for example, regulators like the FAA and EASA give precise testing requirements for critical algorithms.

A Mathematically Rigorous Approach to Venue Compliance

At Aesthetic Integration, we've developed a product, Imandra, that can automatically analyze the design and implementation of financial algorithms to detect regulatory violations. Imandra is powered by recent major advances in the field of 'formal verification.'

To give you a concrete example, we recently published a case study[1] covering this year's settlement between the SEC and UBS ATS. We took the current Form ATS from UBS's website, encoded it in Imandra and demonstrated how issues raised within the settlement can be detected automatically. This includes two major issues raised by the Commission regarding 'sub-penny pricing' and undisclosed crossing constraints.

[1] "Case Study: 2015 SEC Fine Against UBS ATS" is available from www.aestheticintegration.com

While we were writing this letter, a news story came out regarding the SEC's push[2] for more transparency of dark pools. The following proposal applies equally to exchanges and dark pools, and in our opinion, will substantially improve the transparency, safety and stability of our financial markets:

1. Ask venue operators (in this case, IEX) to encode their venue specification (e.g., its order types, conditions for transitions into volatility auctions, delays, etc.) in a mathematically-precise specification language (i.e., a language with a formal semantics) such as the Imandra Modelling Language. In contrast to the English prose in current submissions, this encoding will give you an unambiguous representation of the venue matching algorithm that can then be mathematically analyzed. Using Imandra, for example, the Commission can automatically analyze such a specification for key regulatory requirements.

2. Ask IEX to systematically monitor their production system for conformance to their regulatory submission to ensure that their live venue does not deviate from the design disclosed to the SEC. This can be automated by IEX simply running the (executable) formal design against their daily trading data, checking to see that their production system's behavior agrees with that of the model. Any time they detect a deviation, they should communicate this deviation to the SEC.

3. Use Imandra (or other formal verification tools) to automatically analyze venue designs for potentially unlawful behavior. Our recent white papers "Case Study: 2015 SEC Fine Against UBS ATS" and "Transparent Order Priority and Pricing"[3] have examples of such analysis.

For example, as part of our recent case study on UBS ATS, we encoded in Imandra their Form ATS submission (dated June 1st, 2015). It took us less than a week to do so. Then, we were able to use Imandra to automatically analyze their design for key regulatory requirements with the push of a button.

As exchanges are typically more complex than dark pools, we expect a technical person knowledgeable of the IEX design to need no more than a month to encode a model of their system.

About Formal Verification

We have published a white paper earlier this year, 'Creating Safe and Fair Markets'[4], describing formal verification, how it is currently applied to other industries, and the recent advances that power our application of formal verification to financial markets. In summary, formal verification is an interdisciplinary field of mathematics, computer science and artificial intelligence directed towards analyzing the behavior and implementation of complex algorithms. It is widely relied upon within the US federal government. To list a few examples:

- The FAA requires[5] precise levels of system testing and formal verification within both the Common Criteria Evaluation Assurance Levels and DO-178C[6] frameworks. Safety-critical algorithms such as air

[2] http://www.reuters.com/article/2015/11/17/sec-darkpools-idUSL1N13C1I520151117

[3] "Transparent Order Priority and Pricing" is available from www.aestheticintegration.com

[4] "Creating Safe and Fair Markets" is available from www.aestheticintegration.com

[5] See https://buildsecurityin.us-cert.gov/articles/best-practices/requirements-engineering/the-common-criteria

[6] See http://www.faa.gov/documentLibrary/media/Advisory_Circular/AC_20-115C.pdf

traffic control, onboard autopilots and collision avoidance, and the security of aircraft local area networks must satisfy these rigorous requirements before they are allowed to be deployed.

- The Department of Transportation has commissioned work[7] on creating a formal verification framework for regulating the safety of autopilot algorithms inside self-driving cars and other autonomous vehicles.
- NASA is one of the biggest drivers in the field. Among many other high-profile examples (Mars rovers, etc.), NASA's NextGen Air Traffic Management[8] framework relies on formal verification to ensure its safety.
- The Department of Defense[9] leverages formal verification across numerous applications, including the design and regulation of cryptographic algorithms and secure hypervisors.

With the staggering (and growing) complexity of modern venues and trading systems, we're driven by the fundamental improvements these algorithm analysis techniques will bring to our critical financial infrastructure.

Conclusion

We thank you for the opportunity to comment on IEX's application. In addition to the literature already referenced, you may also wish to consult an internal SEC video recording (in SEC University) of our recent invited lectures at SEC Headquarters (April 6th, 2015).

Sincerely

[illegible] [illegible]

Denis Ignatovich
Co-Founder, AI

Grant Passmore, PhD
Co-Founder, AI

[7] See http://utc.ices.cmu.edu/utc/utc-tset-projects.html

[8] See http://www.hq.nasa.gov/office/aero/asp/airspace/

[9] See http://www.darpa.mil/program/high-assurance-cyber-military-systems

AESTHETIC AI INTEGRATION

CREATING SAFE AND FAIR MARKETS

Creating Safe and Fair Markets

Denis A. Ignatovich and Grant O. Passmore
AESTHETIC INTEGRATION, LTD.
122 Leadenhall St., City of London, EC3V 4AB
www.aestheticintegration.com

Abstract

Many deep issues plaguing today's financial markets are symptoms of a fundamental problem: The complexity of algorithms underlying modern finance has significantly outpaced the power of traditional tools used to design and regulate them. When it comes to exhaustively reasoning about the behaviour of complex algorithms, the only viable solution is formal verification, the use of deep advances in mathematical logic to automatically reason about algorithms and prove properties of programs. Aesthetic Integration is bringing formal verification to financial markets for the first time. In this white paper intended for the wider financial industry, we present our vision for the design and regulation of electronic financial markets empowered by formal verification.

Modern financial markets are built on a staggeringly complex tangle of algorithms. Competitive pressures and economic recession (e.g., decreasing margins and shrinking commission pools) have led to increasingly opaque and unstable markets. The effects of glitches and unfair advantages can be devastating, cratering the confidence of investors and hurting the general public.

In recent years, regulators and the industry have made tremendous progress in defining what safe and fair markets are. What's been missing is a way to analyse and regulate the complex algorithms underlying them.

Flash crashes, questions of fairness and a lack of transparent trading logic within dark pools are all symptoms of a fundamental problem: When it comes to designing and regulating electronic trading systems, financial firms and regulators have not had the right tools for the job.

The solution is formal verification, deep advances in mathematical logic that allow us to automatically reason about algorithms and prove properties of programs. Powered by recent breakthroughs, we can at last scale formal verification to the complex software systems used in financial markets.

Aesthetic Integration's Imandra product is software that brings cutting edge formal verification to the design and regulation of complex financial algorithms. Imandra empowers a broad range of stakeholders — from traders, engineers and compliance officers inside financial firms to economists and enforcement teams inside regulatory agencies — with the proper tools to automatically analyse deep properties of safety, fairness and transparency of critical financial algorithms.

THE BOTTOM LINE: Safety-critical industries already rely upon formal verification to make their algorithms safe. Modern financial markets are safety-critical, too. Now that formal verification technology scales to financial algorithms, the industry and regulators must embrace it.

I. MANAGING THE INFINITE

Real-world financial algorithms are unfathomably complex. A typical trading system may, at any given time, accept hundreds of inputs and compute hundreds of outputs. The set of its possible configurations — its *state space* — is enormous. Faced with such a set of possible scenarios, how can we even begin to grasp

Acknowledgement: We thank Michael Aikins of Chi-X Australia, Austin Gerig of the University of Oxford, Barbara Passmore, and Philip Stafford of the Financial Times for their insightful comments on a draft of this white paper. All errors and omissions that remain are ours alone.

whether a trading system's logic is robust enough to protect itself from making bad decisions? We must find a way to consider *all* possible behaviours of the algorithm to determine *what can possibly go wrong*, and to fix breaches of safety and fairness before they affect markets.

The unprecedented power of formal verification stems from its ability to automatically reason about such enormous state spaces, even *infinitely* large ones. It is quite remarkable, but mathematicians have been reasoning about the behaviour of algorithms over infinite state spaces for a very long time.

To gain some intuition, consider a sorting algorithm

$$F : \text{list } \mathbb{Z} \rightarrow \text{list } \mathbb{Z}$$

that accepts a list of integer values as input and returns as output the input list with its elements sorted in ascending order. How can we *prove* this algorithm will work correctly for all possible inputs? Certainly, we can test F on finitely many cases. But there are an infinite number of possible integer lists. Thus with testing, there will always be some cases (in fact, infinitely many cases) that we miss. Testing gives us no guarantee that bugs do not exist; they may be hidden in difficult to find corner cases not considered by our tests.

With formal verification, we can do (infinitely) better: We can use the proof method of *structural induction* to reason about F over the entire infinite state space induced by the datatypes involved in its execution.

To prove F is correct for all possible inputs, it suffices to prove two properties:

- P_1: The output of F is always sorted.
- P_2: The output of F is always a permutation of its input.

To prove both properties P_1 and P_2, we can use a particular structural induction principle, *list induction*, arguing as follows:

- Base case: P_1 holds of the simplest list.
- Induction step: If P_1 holds for an arbitrary list X, then P_1 will also hold for a new list $(n :: X)$ obtained by prepending an arbitrary integer n to X. Here, both n and X are *symbolic constants*.

If we mathematically prove these two statements, then we have established that the sorting function will work for all possible inputs. With suitable automated theorem proving techniques, the construction of such proofs can often be completely automated. Moreover, if F is buggy (and thus no proof of correctness exists), we can instead automatically derive *counterexamples*, i.e., concrete input values that cause F to fail to meet its specification. Please see the Appendix for a more detailed discussion.

Now contrast this type of rigorous mathematical reasoning with that of presenting several concrete "test cases" for which the function F works and then claiming that, since it works for those few, it should work for all the other infinitely many cases. Such an argument is clearly fallacious. Nevertheless, such "testing" is currently common practice in finance. Its obvious lack of scientific rigour is precisely why systems break down.

To analyse safety and fairness properties of complicated algorithms, we need powerful tools that perform complex mathematical reasoning to prove properties of computer programs automatically. That is, we need the latest advances in formal verification.

Let us first examine formal verification's use in other safety-critical industries. Then we shall discuss how related techniques can empower designers and regulators with the proper tools for ensuring the safety and fairness of algorithms underlying modern electronic financial markets.

II. HOW OTHER INDUSTRIES DEAL WITH COMPLEX ALGORITHMS

From the safety of autopilot systems navigating commercial jets and self-driving cars to the correctness of microchips in mobile phones, companies and governments worldwide rely on formal verification to design and regulate safety-critical hardware and software.

Historically, formal verification has been used most in hardware (e.g., microprocessor) design and aerospace (e.g., autopilot) software safety. With recent advances in automated reasoning, it's become possible to scale automatic formal verification to reason about large-scale software systems. For example, Microsoft now requires device driver code for a piece of hardware to pass Microsoft's formal verification toolchain (the Static Driver Verifier [2]) before the hardware can be "Windows Certified."

Companies like Intel, AMD and Centaur use formal verification in nearly every step of their design process. Much early momentum stems from a major debacle in 1994 when Intel released their Pentium® microprocessor with a bug in its floating point division (FDIV) instruction. A massive recall and subsequent refabrication cost Intel nearly $500,000,000. With the stakes so high, Intel competitor AMD took the pioneering step of engaging formal verification practitioners to verify the correctness of their new K5® processor FDIV design before fabrication, to great success [4]. Today, major hardware companies have large in-house formal verification teams and the technology is integral to their design and development cycles [9, 10].

In aerospace, formal verification is typically used to verify the safety of complex software systems underlying Air Traffic Management and on-board Collision Avoidance for autonomous aircrafts and autopilots. The NASA/NIA formal methods program [1] is one of the leading forces. The aerospace regulatory bodies (FAA in the USA, EASA in Europe) specify use of FV-based ('formal' and 'semi-formal') methods via the DO-178C and Common Criteria software certification levels for safety-critical systems [8, 13]. The US Department of Transportation has recently commissioned related work for autonomous robots and self-driving cars [12].

III. INTRODUCING IMANDRA

Imandra began with our realisation of a deep connection between autopilot and financial algorithms. In fact, we see financial markets as a vast collection of autopilot trading algorithms making critical decisions about transactions constantly. But there is currently a significant divide between the safety of algorithms in aerospace and finance. Our mission is to close this gap — to bring tools that institutions like NASA use for designing safe autopilot algorithms to finance.

But we aim to take formal verification even further. For finance to adopt formal verification, we believe strongly that it must be given a highly automated solution. We aim to give our clients the power of formal verification without requiring them to master the complex mathematics involved.

Formal verification is a vast field, with a diverse collection of techniques designed to address many different classes of problems across a multitude of industries. This immense diversity is often overwhelming, as techniques applicable to one class of problems may fail to work on problems of a (subtly) different nature. Moreover, in order to reason automatically about financial algorithms, new techniques were needed in many areas: nonlinear arithmetic, automated induction, automated model-finding and risk exposure datatypes to name a few.

We designed Imandra from the ground-up specifically for financial algorithms, building upon decades of formal verification research and designing many new proprietary, patent-pending techniques for automated reasoning about financial algorithms. Let us now describe Imandra in more detail.

Imandra models are built using the Imandra Modelling Language (IML). IML is both a high-performance programming language and a "finance-aware" mathematical logic in which properties of IML programs can be stated and proved. Imandra's reasoning engine can be used to construct such proofs, or to compute counterexamples and test-cases automatically.

Imandra has the following key properties:

1. A formal semantics: This allows us to translate any program written in IML into mathematics, i.e., into systems of axioms precisely describing the behaviour of the algorithm. Then, methods of

mathematical proof can be used to reason about the algorithm's behaviour.

2. A high-performance executable semantics: This allows any program written in IML to be compiled into high-performance executable code. In this way, every IML model can be "run" on concrete data. The IML compiler generates efficient code that can then be used directly in production systems. The executable core of IML is an axiomatised subset of the OCaml programming language. Thus, high-performance OCaml tools (compilers, debuggers, etc.) can be brought to bear upon the efficient execution and production use of IML models.
3. Automated reasoning: Powered by Imandra's reasoning engine, deep properties of IML models can be formally proved or disproved automatically. This is made possible by powerful automated theorem proving technology, including many recent advances in SMT, nonlinear decision procedures and model-based automated induction [6, 4, 5]. Imandra's reasoning engine contains many theorem proving algorithms developed specifically for reasoning about fairness and safety properties of trading systems and venues. Moreover, Imandra can automatically derive high-coverage test suites from system specifications.

To ease the modelling of financial computing systems, Imandra is equipped with modelling libraries containing *generic models* of venues, SORs and other trading algorithms. To encode a given venue's matching logic, one need only customise a generic venue model with the business logic specific to the venue of interest. This insulates the user from a significant amount of "boilerplate" modelling. For example, financial constructs such as currencies, asset classes, prices, sector exposures and nonlinear risk attributes of derivatives are provided "out of the box" in IML.

IV. REGULATORY OVERSIGHT

Modern financial institutions have to answer many difficult questions regarding the safety, transparency and fairness of their systems. To address these questions rigorously, the actual algorithms involved must be analysed.

For example: How can a financial intermediary prove that its dark pool will never give preference to an internal client (e.g., an internal trading desk) over an external client (e.g., an investor)? The dark pool must have access to certain client information for each order, e.g., to abide by client-specific constraints. Nevertheless, one must ensure that it is not using that information to change its matching decisions to disadvantage anyone.

With Imandra, concrete fairness principles such as a lack of discriminatory and unlawful use of customer information in pricing decisions can be encoded and analysed for a dark pool automatically. If Imandra proves the dark pool's matching logic fair in this sense, it will construct a mathematical proof that can be independently verified. If Imandra instead finds a counterexample — a scenario in which the matching logic disadvantages a client on price, for example — it will automatically translate this scenario into a sequence of FIX® messages that cause the dark pool to exhibit the unfair behaviour. Such counterexamples are of tremendous value for finding and fixing bugs and violations before they hit the markets.

We believe Imandra (and formal verification more generally) will be of immense value to financial regulators. In this section, we highlight some key applications in the regulatory space. For each application, we present three points: A *problem*, an *immediate solution* and a *long-term vision*. The immediate solutions are important first steps that can already be accomplished with the current features of Imandra, in consultation with regulators and industry. The long-term visions are more speculative and represent our vision for the future of finance.

IV.1 Designing Directives

Problem: Regulators need to design and communicate directives on properties of financial algorithms. As much as possible, these directives need to be precise and unambiguous. Moreover, market participants need seamless ways to incorporate these directives into their design, testing and compliance processes.

Immediate solution: Imandra can be used to encode regulatory principles that are easily expressible in a "finance-aware" mathematical logic (IML). This includes a broad class of directives giving specific quantitative constraints on the allowed behaviour of algorithms underlying trading systems, e.g., ensuring that systems contain appropriate risk limits (e.g., no order is above trader's limits), that orders have maximum size (a system-wide constraint on how big an order may be), or that the system does not sell short a restricted stock. Many fairness regulations fall into this class, such as those restricting the use of customer data in matching and pricing decisions.

Regulators themselves can use Imandra to reason about these encoded constraints, applying Imandra's reasoning engine to determine if certain constraints are satisfied by model trading systems built in IML, or to understand subtle relationships between different directives (does directive A always imply directive B?).

This work can be done in consultation with Aesthetic Integration, with Imandra being enhanced on-demand to support a regulator's needs. Simultaneously, Aesthetic Integration can work in consultation with financial firms, helping apply Imandra to analyse their systems with respect to the formalised regulations.

Long-term vision: In the long-term, formal languages like IML will become the lingua franca of financial regulations and system specifications, and formal verification systems like Imandra will be the "design studio" for understanding the market effects of newly proposed regulations.

Financial firms will provide regulators and their clients with formal models of their trading systems and venues. If regulators wish to understand the market effects of a newly proposed regulation, they will be able to run it against the latest collection of models of market participants, to understand which ones would pass and which ones would fail, and why.

Regulators will provide formalised regulations (and proposed regulations) to the industry and general public. Financial firms will be able to automatically import the latest regulations into their development framework, analysing both their current and prospective systems for compliance automatically. The public will have a precise medium for understanding, analysing and proposing improvements to regulations.

IV.2 Quantifiable Testing Standards

Problem: There is a consensus on the need for testing financial computing systems. Major recent regulatory directives clearly require it. However one crucial detail has been missing — a precise definition of "sufficient testing."

Immediate solution: Armed with a formal model of a financial computing system, the adequacy of a test suite can be analysed in powerful ways. With Imandra, market participants can use formalised regulatory directives and specifications of their systems to generate test suites with unprecedented coverage.

Sophisticated metrics are needed for evaluating the adequacy of an algorithm's test suite. In current (pre-FV) practice, most test suites are written by hand with no mathematical analysis done to determine which aspects of the state space of the system are covered by tests. Bugs hidden deep within complex combinations of system logic are often completely missed by testing, to profound negative effect.

In consultation with Aesthetic Integration, Imandra models of the systems under test can be constructed, and our test suite analysis and generation machinery can be customised and integrated into a firm's development process. Market participants can then test their production systems in far more thorough and cost effective ways. Institutions can set quantifiable testing standards and actually enforce them.

Long-term vision: In the long-term, regulators will themselves set quantifiable standards of testing for each of their regulatory principles.

For example, in a recent Aesthetic Integration case study analysing a simple exchange matching logic, we've shown that more than 400,000 separate components of the (infinite) state space of the venue model must be analysed to determine whether the matching logic satisfies a particular fairness condition (pertaining to the non-use of client information in match pricing).

For each formalised regulatory principle, regulators will be able to set minimum testing standards for production systems. Firms will be able to import these automatically, use tools like Imandra to generate such test suites, and send the resulting test results to regulators for automated analysis.

IV.3 Linking Regulation With Financial Economics

Problem: Regulators must have a feedback loop between their directives and the overall performance of the financial markets. They must evaluate whether participants' algorithms have been constrained too much or not enough. If the algorithms are over-constrained, little trading takes place and the markets do not perform their ultimate function of transferring capital and ownership between their participants. If algorithms are under-constrained, then markets exhibit events such as the "flash crash" and recurring concerns of unlawful exploitation of microstructure effects.

Immediate solution: The issues of interactions of numerous concurrent systems are not unique to financial markets. Hardware manufacturing firms rely on formal verification to reason about possible sequences of *concurrent* events that would lead the system to violate requirements.

In similar fashion, Imandra may be used to reason about the behaviour of a finite collection of trading algorithms interacting via venues. In consultation with Aesthetic Integration, regulators can use Imandra to design a "sandbox" of models of various trading strategies and venues, and to analyse ("abduct") which constraints on the algorithms and venues would prevent certain classes of bad events. For example, one may wish to avoid a sudden drop in market prices driven by trading algorithms trying to "outrun" each other. With Imandra, financial economists are empowered with tools to undertake this research.

Long-term vision: Our vision for the financial markets is to have both participant firms create formal specifications of their systems, and for the regulators to have formal specifications of their regulatory directives. With such an ecosystem, formal verification will be used to provide full decision attribution analysis. Regulators will be able to pinpoint exactly which elements of trading logic (or lack thereof) led to specific economic events under study. For example, the logic responsible for creating/amending orders during events of extreme market volatility will be quickly isolated using both market data and formal models of the systems involved.

Joining formal algorithm specifications with CAT-like data [14] will help close the feedback loop between analysis of economic events and development of regulatory directives. This will allow systematic calibration of market microstructure regulations for the right trade-off between transaction volume and stability.

IV.4 Demonstrating Compliance (in an IP-aware manner)

Problem: Financial firms need to demonstrate to regulators the compliance of their systems. This is currently costly with much undesirable imprecision. Intellectual property concerns further complicate this process.

Immediate solution: Using Imandra, financial institutions can formalise the most critical components of their algorithmic systems, e.g., the matching logic of a venue or the risk gate component of an SOR. In consultation with Aesthetic Integration, key regulatory directives and internal risk requirements can be formalised in IML and the system specifications can be subjected to Imandra's automated formal verification.

Long-term vision: In the long-term, formal verification will simplify many time consuming and expensive compliance functions. For example, consider the process of proving to a regulator that a dark pool is compliant. Provided with formal specifications of the systems and regulations, tools like Imandra will generate the following:

- For each formally specified regulatory directive, a mathematical proof that the system specification is compliant. Such proofs are expressed in a formal mathematical logic and can be verified independently by a third party.
- Test suite coverage metrics and their results.

On the regulatory side, such reports will be processed and verified automatically. Moreover, precise documentation of a system's business logic will be automatically generated from the formal model.

Intellectual property (IP) issues currently pose a challenge for the regulators. Compromising trade secrets guiding the logic of trading systems can expose firms to adverse selection and hurt business.

With formal verification, this issue can be sidestepped in a compelling way: From the regulators' perspective, trade secrets and sensitive IP particular to a high-performance system implementation are irrelevant, provided these systems abide by regulatory constraints. With formal verification, financial firms can demonstrate:

- That they have an internal and formal specification of their system,
- That they have formal mathematical proofs that the specification meets directives on safety and fairness provided by the regulators (also encoded as mathematical objects),
- That they use the formal specification to produce extensive test suites with appropriate coverage metrics and that their implementation of the specification successfully passes the tests.

All of this can be done without releasing particular sensitive details of their production implementation. Moreover, comprehensive documentation of the algorithm's business logic can be produced automatically from the specification when appropriate. If issues are later found in the production implementation (issues that were not caught with the high-coverage test suites), then the formal specification can be used to pinpoint these issues and drive fixes. Compared to current practices, this gives regulators and financial firms a far more precise framework for reasoning about the compliance of complex IP-laden production systems. Of course, if desired, IP-sensitive details of production systems can be subjected to formal verification as well, e.g., through proving equivalence of a high-performance, low level algorithm used in production with its low-performance, high-level specification.

CLOSING REMARKS

Our mission is to provide financial markets and regulators with powerful tools for managing the complex algorithms underlying modern trading systems and venues. Imandra by Aesthetic Integration brings revolutionary advances in formal verification to bear on financial algorithms, at last allowing us to scale robust engineering methods used in other safety-critical industries to finance.

We are driven by the fundamental improvements these latest advances will bring to global financial markets. Formal verification will eliminate significant portions of the costs and resources required to operate and regulate trading businesses. Precision and systematic rigour will replace ambiguous and ad hoc approaches to managing complicated trading systems.

Imandra will help you build safer, more stable and compliant businesses. Together let's make financial markets safe and fair.

REFERENCES

[1] The NASA Formal Methods Research Program: http://ti.arc.nasa.gov/nfm/.

[2] Thomas Ball, Ella Bounimova, Vladimir Levin, Rahul Kumar, and Jakob Lichtenberg. The static driver verifier research platform. In *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings,* pages 119–122, 2010.

[3] Robert S. Boyer and J Strother Moore. *A Computational Logic Handbook.* Academic Press Professional, Inc., San Diego, CA, USA, 1988.

[4] Bishop Brock, Matt Kaufmann, and J Strother Moore. ACL2 theorems about commercial microprocessors. In Mandayam Srivas and Albert Camilleri, editors, *Formal Methods in Computer-Aided Design, volume 1166 of Lecture Notes in Computer Science,* pages 275–293. Springer Berlin Heidelberg, 1996.

[5] Leonardo de Moura and Nikolaj Bjørner. Satisfiability modulo theories: Introduction and applications. *Communications of the ACM,* 54(9):69–77, September 2011.

[6] Leonardo de Moura and Grant Olney Passmore. The Strategy Challenge in SMT Solving. Chapter in the book *Automated Reasoning and Mathematics: Essays in Memory of William W. McCune*., Maria Paola Bonacina and Mark Stickel, editors, pages 15–44. Springer-Verlag, Berlin, Heidelberg, 2013.

[7] Leonardo de Moura and Grant Olney Passmore. Computation in real closed infinitesimal and transcendental extensions of the rationals. In Maria Paola Bonacina, editor, CADE-24, volume 7898 of *LNCS*. Springer, 2013.

[8] Gabriella Gigante and Domenico Pascarella. Formal methods in avionic software certification: The DO-178C perspective. In *Proceedings of the 5th International Conference on Leveraging Applications of Formal Methods, Verification and Validation: Applications and Case Studies - Volume Part II,* ISoLA'12, pages 205–215, Berlin, Heidelberg, 2012. Springer-Verlag.

[9] Warren A. Hunt, Sol Swords, Jared Davis, and Anna Slobodova. Use of Formal Verification at Centaur Technology. In David S. Hardin, editor, *Design and Verification of Microprocessor Systems for High-Assurance Applications,* pages 65–88. Springer US, 2010.

[10] Roope Kaivola, Rajnish Ghughal, Naren Narasimhan, Amber Telfer, Jesse Whittemore, Sudhindra Pandav, Anna Slobodová, Christopher Taylor, Vladimir Frolov, Erik Reeber, and Armaghan Naik. Replacing Testing with Formal Verification in Intel Core I7® Processor Execution Engine Validation. In *Proceedings of the 21st International Conference on Computer Aided Verification,* CAV '09, pages 414–429, Berlin, Heidelberg, 2009. Springer-Verlag.

[11] Matt Kaufmann, J Strother Moore, and Panagiotis Manolios. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[12] Stefan Mitsch, Grant Olney Passmore, and André Platzer. Collaborative verification-driven engineering of hybrid systems. *Mathematics in Computer Science,* 8(1):71–97, 2014.

[13] Special Committee of RTCA. DO-178C, software considerations in airborne systems and equipment certification, 2011.

[14] US Securities and Exchange Commission. Rule 613: Consolidated Audit Trail. https://www.sec.gov/divisions/marketreg/rule613-info.htm.

APPENDIX: INDUCTIVE PROOFS OVER DATATYPES AND ALGORITHMS

I. Mathematical Induction

Consider a computable monadic predicate

$$P : \mathbb{N} \to \{True, False\}$$

on the natural numbers. For any given $n \in \mathbb{N}$, $P(n)$ is either True or False. Imagine we wish to prove that $P(n)$ holds for *all* $n \in \mathbb{N}$, i.e., $\forall x \in \mathbb{N}(P(x))$. How can we do it? One powerful proof method is given by the principle of *mathematical induction*:

$$\frac{P(0) \qquad \forall x \in \mathbb{N}\,(P(x) \implies P(x+1))}{\forall x \in \mathbb{N}(P(x))}$$

This says: If we can prove both $P(0)$ and $\forall x \in \mathbb{N}\,(P(x) \implies P(x+1))$, then we can conclude $\forall x \in \mathbb{N}(P(x))$. Why is this principle true?

Assume the hypotheses $P(0)$ and $\forall x \in \mathbb{N}\,(P(x) \implies P(x+1))$. We can derive a contradiction from the existence of a counterexample to the conclusion $\forall x \in \mathbb{N}(P(x))$, i.e., from the assumption $\exists n \in \mathbb{N}(\neg P(n))$. Assume there exists such a counterexample. By well-foundedness of $<$ on $\mathbb{N}$, there exists a least $w \in \mathbb{N}$ s.t. $\neg P(w)$. By our assumption of $P(0)$, we know $w > 0$. But then $w-1 \in \mathbb{N}$ and thus $P(w-1)$. By our assumption of $\forall x \in \mathbb{N}\,(P(x) \implies P(x+1))$, we know that $P(w-1) \implies P(w)$. But then $P(w)$ holds, which is a contradiction. Thus, $\forall x \in \mathbb{N}(P(x))$ must hold. In this way, we see we can derive the principle mathematical induction from the well-foundedness of the standard strict ordering relation ($<$) on $\mathbb{N}$.

However, from the perspective of computation, there is another, even more direct way to derive the induction principle for $\mathbb{N}$: By observing that $\mathbb{N}$ is an inductively generated datatype.

Consider the following IML definition of a datatype nat of natural numbers:

```
type nat = Zero | S of nat
```

This definition says that a value x is a nat iff $x = \texttt{Zero}$ or $x = \texttt{S}(y)$ where y is a `nat` (S as in "successor"). We say `nat` has two constructors: Zero and S. Moreover, Zero is a "base" constructor, while S is an "inductive" one. For example, the following are both values of type `nat`:

```
Zero : nat
S(S(S(Zero))) : nat
```

The inductive generation of the datatype guarantees something very important: That there exist no ways of constructing a value of type `nat` other than through these two constructors. This gives us a direct method for justifying the following *structural induction* principle, obviously isomorphic to the principle of mathematical induction given above:

$$\frac{P(\texttt{Zero}) \qquad \forall x : \texttt{nat}\,(P(x) \implies P(\texttt{S}(x)))}{\forall x : \texttt{nat}(P(x))}$$

It is easy to see how this principle can be derived mechanically from the definition of the datatype. To gain some intuition for induction in general, let us use mathematical induction to prove a simple theorem often credited to Gauss.

Theorem 1 (Gauss).

$$\forall n \in \mathbb{N} \left(\sum_{i=0}^{n} i = \frac{n(n+1)}{2} \right).$$

Proof.

Let $P(n)$ denote the statement $\left(\sum_{i=0}^{n} i = \frac{n(n+1)}{2}\right)$. We shall prove $\forall n \in \mathbb{N}(P(n))$.

Base case: $P(0)$ is immediate as $\sum_{i=0}^{0} i = \frac{0(0+1)}{2} = 0$.

Induction step: Assume $P(n)$, i.e., $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$ for some $n \in \mathbb{N}$. By mathematical induction, it then suffices to prove $P(n+1)$:

$$\sum_{i=0}^{n+1} i = \frac{(n+1)(n+2)}{2}.$$

Let us calculate. Note our use of our assumption $P(n)$ to replace $\sum_{i=0}^{n} i$ with $\frac{n(n+1)}{2}$:

$$\sum_{i=0}^{n+1} i = \left(\sum_{i=0}^{n} i\right) + (n+1) = \frac{n(n+1)}{2} + (n+1) = \frac{n^2+3n+2}{2} = \frac{(n+1)(n+2)}{2}.$$

Using automated theorem proving technology, Imandra can prove this theorem automatically. Moreover, if we have errors in our theorem statements or function definitions, Imandra can help us find and fix these errors by automatically deriving relevant counterexamples.

II. List and Tree Induction

Consider now a datatype of lists of values of type α, where α is arbitrary. In IML notation, we can represent this type as follows:

```
type 'a list = Nil | Cons of 'a * 'a list
```

For example, the following are concrete lists of int and string values, respectively:

```
[-7;2;3;5] = Cons(-7, Cons(2, Cons(3, Cons(5, Nil))))) : int list
["a";"b";"c"] = Cons("a", Cons("b", Cons("c", Nil))) : string list
```

A structural induction principle for proving universal theorems over lists is as follows:

$$\frac{P(\texttt{Nil}) \qquad \forall x : \alpha\ \texttt{list}\ \forall n : \alpha\ (P(x) \implies P(\texttt{Cons}(n, x)))}{\forall x : \alpha\ \texttt{list}(P(x))}$$

To illustrate list induction, let us prove that the following simple append function is associative:

```
append(Nil,y) = y
append(Cons(x,xs),ys) = Cons(x,append(xs,ys))
```

Theorem 2. $\forall xyz\ (\texttt{append}(x, \texttt{append}(y, z)) = \texttt{append}(\texttt{append}(x, y), z))$.

Proof. By induction on x. Let $P(x)$ denote $\forall yz\ (\texttt{append}(x, \texttt{append}(y, z)) = \texttt{append}(\texttt{append}(x, y), z))$.

Base case: Show P(Nil). By definition of append, $P(\texttt{Nil}) = \forall yz\ (\texttt{append}(y,z) = \texttt{append}(y,z))$, which is obviously true.

Induction step: Assume $P(\texttt{X})$ (the "Induction Hypothesis"), and show $P(\texttt{Cons}(\texttt{c}, \texttt{X}))$. Then,

$$\begin{aligned} \texttt{append}(\texttt{Cons}(\texttt{c},\texttt{X}), \texttt{append}(y,z)) &= \texttt{Cons}(\texttt{c}, \texttt{append}(\texttt{X}, \texttt{append}(y,\ z))) && \textit{by def} \\ &= \texttt{Cons}(\texttt{c}, \texttt{append}(\texttt{append}(\texttt{X}, y),\ z)) && \textit{by IH} \\ &= \texttt{append}(\texttt{Cons}(\texttt{c}, \texttt{append}(\texttt{X}, y)), z) && \textit{by def} \\ &= \texttt{append}(\texttt{append}(\texttt{Cons}(\texttt{c}, \texttt{X}), y), z) && \textit{by def} \end{aligned}$$

where def is the definition of append and IH is the Induction Hypothesis.

Indeed, this is a trivial automatic proof. However, this simple example illustrates an important point: Care must be taken when choosing "how" one performs a proof by induction. This problem was solved by performing list induction on x. But what if we had made a "wrong" choice and attempted to do list induction on y? When reasoning about nontrivial algorithms, one often needs powerful induction heuristics for constructing the "right" instances of the relevant induction principles. Besides carefully selecting the right (combination of) variable(s) upon which to do induction, one often also needs to "generalise" the theorem being proved in order for the induction step to hold. When automatically reasoning about financial algorithms, in addition to powerful techniques for inductive proof, one also needs powerful decision procedures for many forms of linear and nonlinear arithmetic, boolean logic, theories of bit-vectors and arrays and datatypes for representing risk exposures.

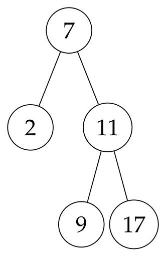
As a final example of structural induction, let us consider a datatype of binary trees defined as follows:

```
type 'a tree = Leaf of 'a | Tree of 'a * 'a tree * 'a tree
```

For example, the following is such a tree:

```
Tree(7, Leaf(2), Tree(11, Leaf(9), Leaf(17))) : int tree
```

which might be visualised as:



From the definition of the datatype, we can derive the following principle of (e.g., integer) tree induction:

$$\frac{\forall x : \mathtt{int}(P(\mathtt{Leaf}(x))) \qquad \forall x, y : \mathtt{int\ tree}\ \forall n : \mathtt{int}\ (P(x) \wedge P(y) \implies P(\mathtt{Tree}(n, x, y)))}{\forall x : \mathtt{int\ tree}(P(x))}$$

As an exercise, an interested reader might try to prove the following theorem by tree induction:

$$\mathtt{num\_nodes}(T) \leq 2^{\mathtt{height}(T)+1} - 1,$$

where num_nodes and height, both of type (α $\mathtt{tree} \rightarrow \mathtt{nat}$), are defined in the natural way.

III. More Powerful Forms of Induction

Though structural induction is often powerful enough for the analysis of financial algorithms, there are times when more sophisticated induction principles are needed. One powerful method is that of *recursion induction*. Beyond this, the most general form of induction is that of *well-founded induction*. The setting for well-founded induction is the *ordinals*. Ordinals are equivalence classes of well-orderings. In set theory, we usually represent an ordinal by a canonically chosen representative, using an encoding due to von Neumann (the "von Neumann ordinals"). In this encoding, 0 is represented by $\emptyset$, and the successor of a von Neumann ordinal a is given by $\alpha \cup \{\alpha\}$.

The standard strict ordering < on $\mathbb{N}$ is called ω (with < encoded as $\in$), and is given as follows:

$$\omega = \langle \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}, \ldots\}, \in \rangle \quad \equiv \quad \langle \{0, 1, 2, 3, \ldots\}, < \rangle.$$

Let *On* be the class of ordinals. Every $\alpha \in On$ can be uniquely represented in the form

$$\omega^{\beta_1} c_1 + \omega^{\beta_2} c_2 + \ldots + \omega^{\beta_k} c_k \qquad .$$

where the arithmetical operations are those for ordinal arithmetic, given shortly. Every ordinal α is either a *successor* ordinal, i.e., $\exists \beta \in On(\alpha = \beta + 1)$ or a *limit* ordinal, the supremum of the set of smaller ordinals. For example, 3 and $2\omega + 7$ are successor ordinals, while ω and ω^{ω} are limits.

Ordinal Arithmetic:

$$\alpha + 0 = \alpha$$

$$\alpha + (\beta + 1) = (\alpha + \beta) + 1$$

$$\alpha + \beta = lim_{\delta \to \beta} (\alpha + \delta) \quad \text{when } \beta \text{ a limit ordinal}$$

$$\alpha \cdot 0 = 0$$

$$\alpha \cdot (\beta + 1) = (\alpha \cdot \beta) + \alpha$$

$$\alpha \cdot \beta = lim_{\delta \to \beta} (\alpha \cdot \delta) \quad \text{when } \beta \text{ a limit ordinal}$$

$$\alpha^0 = 1$$

$$\alpha^{\beta+1} = \alpha^{\beta} \cdot \alpha$$

$$\alpha^{\beta} = lim_{\delta \to \beta} \left(\alpha^{\delta} \right) \quad \text{when } \beta \text{ a limit ordinal}$$

Induction principle:

$$\frac{\forall a \in A\, [\forall b \in B\, (b < a \Longrightarrow P(b))] \Longrightarrow P(a)}{\forall a \in A(P(a))}$$

Many powerful automated reasoning techniques exist for well-founded induction, especially those due to Boyer-Moore [3] and found within the ACL2 theorem prover (for quantifier-free induction up to the ordinal ϵ_0) [11]. With recent advances in automated model construction, these techniques can be significantly strengthened, e.g., by using (non-standard) counterexamples to guide nuanced forms of inductive generalisation. Imandra's automated induction builds upon these many advances.

ABOUT AESTHETIC INTEGRATION

Aesthetic Integration Limited is a financial technology company based in the City of London. Created by leading innovators in software safety, trading system design and risk management, AI's patent-pending formal verification technology is revolutionising the safety, stability and transparency of global financial markets.

Imandra, the world's first formal verification solution for financial markets:

- Radically reduces costs associated with trading system design, implementation and compliance
- Analyses correctness and stability of system designs for regulatory compliance (MIFID II and RegSCI)
- Uncovers non-trivial system design flaws and bugs in implementation
- Creates exhaustive test suites with quantifiable state-space coverage metrics

Founders

DENIS IGNATOVICH has nearly a decade of experience in trading, risk management, quantitative modeling and complex trading system design at a leading global investment bank.

He holds degrees in Computer Science and Finance from the University of Texas at Austin and an MSc in Finance from the London School of Economics.

GRANT PASSMORE has ten years' industrial formal verification experience, and has been a key contributor to safety verification of algorithms at Cambridge, Carnegie Mellon, Edinburgh, Microsoft Research and SRI.

He earned his PhD from the University of Edinburgh and is a Life Member of Clare Hall, University of Cambridge.

See our website for details: www.aestheticintegration.com

Contact

Contact@AestheticIntegration.com
122 Leadenhall Street | City of London
EC3V 4AB | United Kingdom | +44 20 3773 6225

AESTHETIC AI INTEGRATION

CASE STUDY: 2015 SEC FINE AGAINST UBS ATS

Case Study: 2015 SEC Fine Against UBS ATS

Denis A. Ignatovich and Grant O. Passmore
Aesthetic Integration, Ltd.

Abstract

This report forms part of AI's application into the UBS Future of Finance Challenge (Banking Efficiency Challenge)[1].

This year's $14 million settlement[2] *between the US Securities and Exchange Commission (SEC) and UBS over allegations of misconduct in design, marketing and implementation of their ATS (dark pool) highlights the financial services industry's ongoing struggles with the staggering (and growing) complexity of trading algorithms.*

We demonstrate how UBS can leverage AI's groundbreaking formal verification technology to prevent further regulatory fines related to the design and implementation of UBS dark pools. Powered by latest scientific breakthroughs, our product Imandra is able to automatically prove properties of fairness and best execution of venue designs and test production implementations with unprecedented rigour. We demonstrate how Imandra can automatically detect and test for key recent issues raised by the SEC.

Furthermore, based on UBS's publicly available Form ATS filing, we apply Imandra to highlight additional potential issues[3] *with UBS's current dark pool design.*

Finally, we discuss applications of Imandra to a wide range of financial algorithms, including routing systems and smart contracts.

[1] https://innovate.ubs.com/
[2] http://www.sec.gov/news/pressrelease/2015-7.html
[3] This case study is based solely on the publicly available SEC documents and UBS Form ATS (dated June 1st, 2015).

Changing The Process

In this report, we showcase our Imandra algorithm analysis technology by applying it to the recent $14mm settlement between UBS and the SEC and analysing the design of UBS ATS (as described in the publicly available Form ATS dated June 1st, 2015) with respect to issues raised in the SEC Order. In addition, we use Imandra to highlight some additional potential issues in the current design of UBS ATS.

Before we dive into the technical details, let us say a bit about Imandra and how it radically improves the process of trading system design, delivery and regulation. At its core, Imandra empowers a broad range of stakeholders with the ability to ask deep questions about an algorithm's possible behaviours, to verify designs for safety, fairness and regulatory principles, and to analyse implementations for conformance to their design[4].

The SEC Order contains several quotes from UBS employees highlighting internal challenges in designing and implementing the dark pool. Here's one taken from page 10:

> *"If we confirm this pricing decision came from PTSS classic," he wrote, "can we not spend to[o] much time on research – we know classic has this issue, its being phased out, and we have dug through examples – to[o] many times already."*

As we illustrate below, Imandra is more than a tool for fixing bugs in software. Imandra is a business tool connecting various stakeholders responsible for the process of designing and delivering trading systems. By using Imandra, businesses optimise their costs, while effectively managing technology and regulatory risks.

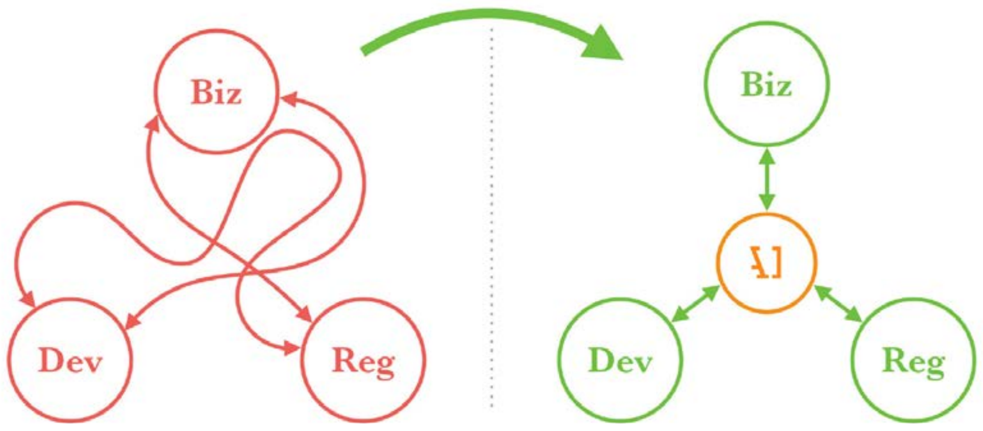


FIGURE 1: IMANDRA TRANSFORMING THE PROCESS OF CREATING TRADING SYSTEMS

In a typical investment bank, the process of designing, implementing and regulating trading systems requires the collaboration of many players with a diverse collection of expertise. Despite their different perspectives, they all require tools for the *analysis of algorithms*. Fundamentally, trading algorithms have become too complex to analyse by hand. Imandra brings the hard science of *formal verification* to analyse algorithms and radically improves the overall process.

[4] Please see our white papers "Creating Safe And Fair Markets" and "Transparent Order Priority and Pricing" available at www.aestheticintegration.com for more background on Imandra.

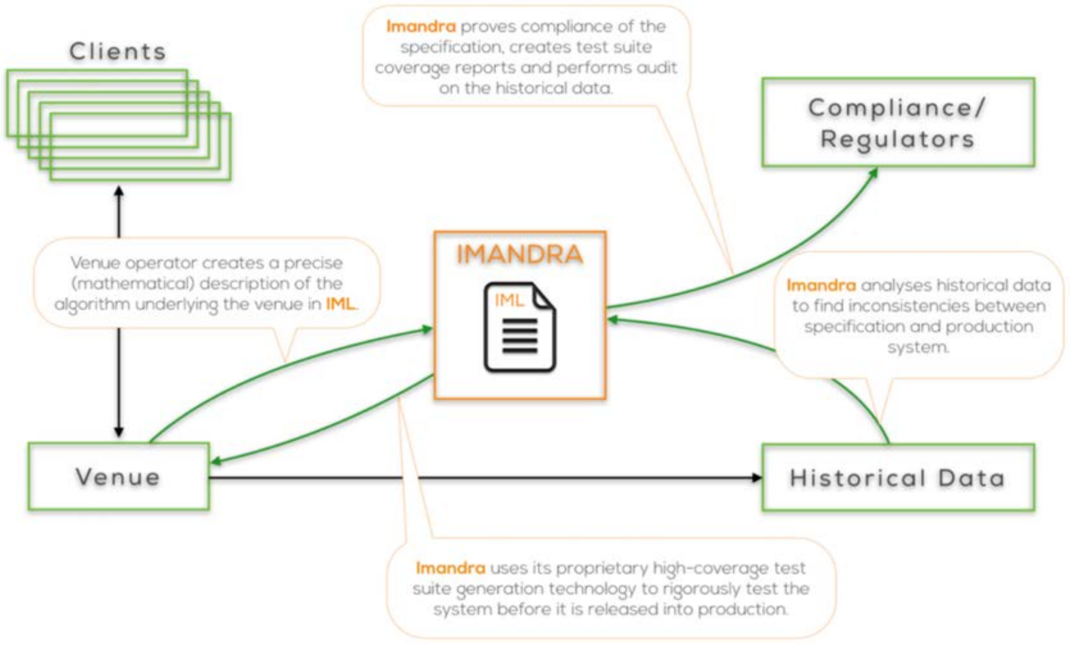

Clients
Imandra proves compliance of the specification, creates test suite coverage reports and performs audit on the historical data.
Compliance/ Regulators
Venue operator creates a precise (mathematical) description of the algorithm underlying the venue in IML.
IMANDRA
IML
Imandra analyses historical data to find inconsistencies between specification and production system.
Venue
Historical Data
Imandra uses its proprietary high-coverage test suite generation technology to rigorously test the system before it is released into production.

FIGURE 2: AUTOMATING COMPLIANCE WITH IMANDRA

Business: With Imandra, the business has a complete and precise design that can be queried and analysed automatically. This is similar to how an architect does not have to personally inspect every floor of a building she designed to understand how many rooms there are. Business stakeholders can use Imandra to immediately understand side-effects (including regulatory impact) of additional features such as new order types or client-specific constraints, BEFORE development starts and systems go into production.

Technology: With Imandra, venue developers can have a precise specification of system functionality. This in turn cuts down time needed to understand business requirements. Quality Assurance teams gain tremendous power and efficiency with Imandra's automated test suite generation enumerating important logical corner cases. Those responsible for systems that send orders to venues can query the Imandra specification to answer key questions about how the venue will communicate with their system. This is a radical improvement over current industry practice, e.g., the error-prone manual deciphering of ambiguous PDF documents and marketing materials.

Regulatory functions (compliance officers): With Imandra, compliance officers can encode and enforce regulatory directives and have full oversight of the regulatory status of the trading system design and implementation.

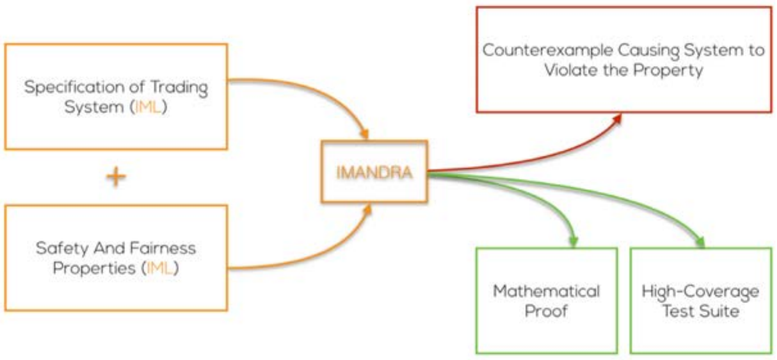

Specification of Trading System (IML)
+
Safety And Fairness Properties (IML)
IMANDRA
Counterexample Causing System to Violate the Property
Mathematical Proof
High-Coverage Test Suite

FIGURE 3: IMANDRA OVERVIEW

With Imandra, businesses can optimise the costs and time they commit to making changes to their venue designs. Regulators can automate analysis of the effects of modifications to venue designs and create a systematic approach to regulating venues. Those designing and implementing systems (e.g., SOR's) that connect to the venues can at last have unambiguous descriptions of how those venues operate.

The Roadmap

The SEC Order

The SEC Order describes several issues regarding the design and operation of the UBS dark pool (in the US) from 2008 to 2012. At a high-level, the SEC raised two main complaints:

1. 'Sub-penny' pricing - functionality within the venue to process order prices with increments less than the statutory minimum. The Order claims there were two reasons for such functionality:
   - A. Two order types that specifically allowed for this behaviour and were not disclosed to all clients of the venue and the regulators.
   - B. Implementation ('technical') errors on behalf of the venue and the internal Smart Order Router (SOR) system that submitted invalidly priced orders to the venue.
2. An undocumented feature constraining matching of internal (originating within the algorithmic trading business) order flow with outside 'non-natural' order flow from market makers ('liquidity providers').

We view these issues in a wider context of headline-making technical glitches and questions regarding venue transparency within dark pools and exchanges. In our view, a significant portion of these problems is due to the financial industry's lack of modern tools for rigorous and scientifically-based analysis of financial algorithms.

Using the SEC Order containing the settlement details and the latest UBS Form ATS, we demonstrate how Imandra can significantly improve the designing, implementing and regulating of modern trading systems and venues. With Imandra, firms like UBS can leverage major scientific breakthroughs to help ensure their venues do not violate regulatory directives, and provide a fully transparent trading experience to its clients.

Imandra and Formal Verification

The issues raised by the SEC are symptoms of a fundamental problem: The complexity of financial algorithms has significantly outpaced the power of traditional tools used to design and regulate them.

Finance is not alone in dealing with complex algorithms. For example, microprocessor designs and autopilot algorithms are also complex. But the hardware and avionics industries have long realised that the state spaces of their safety-critical systems are too complex to understand by hand, and that computer-based *formal verification* techniques must be used to automatically reason about their possible behaviours. Formal verification now plays a crucial role in both hardware and avionics processes for designing safety-critical systems. Regulators like the FAA and the EASA require the use of rigorous mathematically-based methods for demonstrating the safety of autopilot systems before they're allowed to be deployed.

Imandra's patent-pending technology brings formal verification to financial algorithms for the first time.

With four simple steps, UBS can apply Imandra to eliminate significant risks surrounding its dark pool:

1. Encode the matching logic (i.e., as given in Form ATS) in the Imandra Modelling Language (IML). This allows Imandra to *reason* about the possible behaviours of the venue, providing designers, developers, testers and regulators with the ability to *query* the trading system design for key properties of interest ("is it ever possible for the matching algorithm to violate the following principle?"). Moreover, this encoding is very easy to do. As discussed below, we have built a full-featured Imandra model of the UBS dark pool based upon the publicly available Form ATS document dated June 1st, 2015. This takes only ~800 lines of IML code.
2. Encode properties of the model you wish to reason about in IML. Imandra will process them to verify that the trading system design is compliant with regulatory directives (e.g., that it does not admit sub-penny pricing or unlawful prioritisation of orders). We give examples below.
3. Based on the logic of the model, use Imandra's proprietary Test Suite Generation (TSG) technology to generate high-coverage test suites to ensure production systems are thoroughly tested for conformance to their verified design and documentation.
4. Use Imandra to compile a high-performance venue simulator and use it to automatically audit historical data created by the dark pool. Such automated audits provide live monitoring and deep analysis of the performance of the dark pool, ensuring that its behaviour is consistent with its design, documentation and marketing materials.

These four steps will result in a vastly more thorough and tight governance process around designing and running the dark pool. Moreover, it will save UBS considerable time and money.

Verification Goals

We refer to the properties we wish to verify about system designs as *verification goals* (VGs). This report will describe four such goals. The first two goals are motivated by the SEC Order. The fourth comes from our proprietary set of verification goals we developed to help our clients meet Regulation SCI and MIFID II requirements. The third is an interesting discovery Imandra made as we encoded the model.

We shall consider the following verification goals:

1. *"Sub-Penny" Pricing* - will the venue accept prices in increments less than the tick size?

2. *Crossing Constraints* - for a typical dark pool, there are many valid reasons why two orders will not trade with each other, even if their prices are compatible. However, there can also be invalid and illegal reasons for blocking a match. Just looking at the post-trade data will make it very difficult to find these issues. With Imandra, you can easily ensure that the venue will not illegally prohibit any two orders from trading with each other.
3. *Transitivity of Order Ranking* - when sorting a list of items (e.g., lists of integers, or orders within an order book, etc.), it is critical that the comparison function used to rank items is *transitive*. For example, the normal "greater-than" relation (">") on integers is transitive: if (a > b) and (b > c), then it always follows that (a > c). Because > is transitive, we can use it to sort a list of integers and receive a sensible output. When a comparison function takes a more complicated form, such as an `order_higher_ranked` function used when sorting orders in an order book, we must be careful to ensure that transitivity still holds. In case it doesn't, then sorting based upon that order ranking may give inconsistent and unpredictable results. Because of the noise and sheer amount of transaction data, such issues are nearly impossible to isolate by looking at post-trade data alone. Imandra analyses the design of the order sorting logic directly.

We examine the order priority logic described in the UBS Form ATS, and show one way it implies that the order ranking function is not transitive. Moreover, Imandra automatically derives concrete scenarios that illustrate the transitivity violation.

4. *Order Priority* - the US market microstructure is filled with numerous order types. They may have different attributes and provide a tailored trading experience. But ultimately they must abide by common regulatory requirements. One such requirement is that no order may 'jump the queue'.

Creating an Imandra Model of UBS ATS

Our complete Imandra model of UBS ATS (as described in the referenced Form ATS) is roughly 800 lines of IML code. In terms of the workload involved in creating it, we expect it should not take more than two weeks for a person familiar with the actual specification of the venue. Because most venues share much in common with each other in that they must maintain sorted order books, match orders, send back fills, etc., Imandra comes equipped with "generic models" of venues. This allows one to quickly develop a specific venue model by only customising aspects that are particular to that venue.

Our UBS model includes the following high-level components:

Order Types

Section 2.2 of the Form ATS declares the following:

"Order Types:

- Pegged Orders (both Resident and IOC TimeInForce). Pegging can be to the near, midpoint, or farside of the NBBO. Pegged Orders may have a limit price.
- Limit Orders (both Resident and IOC TimeInForce)
- Market Orders (both Resident and IOC TimeInForce)

Conditional Indication Types:

- Pegged Conditional Indications (Resident TimeInForce only). Pegging can be to the near, midpoint,or far side of the NBBO. Pegged Conditional Indications may have a limit price.
- Limit Conditional Indications (Resident TimeInForce only)"

Our first task is to define explicitly all of the different order types allowed in the venue. Figure 4 shows the IML definitions for order types.

```
type order_type = MARKET | LIMIT | PEGGED | PEGGED_CI | LIMIT_CI
```

FIGURE 4: DECLARATION OF THE ORDER TYPES SUPPORTED BY THE ATS

Other parts of the IML model will assign operational meaning to these order types. Here we explicitly state the 5 types of orders that the venue supports. One of the great advantages of using Imandra is that it forces users to be precise. For example, Imandra will not accept a model as complete unless the user describes how orders are priced for each of declared order type.

Here's an example of the code that calculates the effective price at which an order may trade:

```
let effective_price (side, o, mkt_data) =
  let calc_pegged_price =
    ( match o.peg with
      | FAR -> lessAggressive(side, o.price,
                              (if side = BUY then mkt_data.nbo else mkt_data.nbb))
      | MID -> lessAggressive(side, o.price, mid_point(mkt_data))
      | NEAR -> lessAggressive(side, o.price,
                               (if side = BUY then mkt_data.nbb else mkt_data.nbo))
      | NO_PEG -> o.price )
  in
  let calc_nbbo_capped_limit =
    ( if side = BUY then lessAggressive(BUY, o.price, mkt_data.nbo)
      else lessAggressive(SELL, o.price, mkt_data.nbb) )
  in
  match o.order_type with
  | LIMIT -> calc_nbbo_capped_limit
  | MARKET -> if side = BUY then mkt_data.nbo else mkt_data.nbb
  | PEGGED -> calc_pegged_price
  | PEGGED_CI -> calc_pegged_price
  | LIMIT_CI -> calc_nbbo_capped_limit
  | FIRM_UP_PEGGED -> calc_pegged_price
  | FIRM_UP_LIMIT -> calc_nbbo_capped_limit
```

FIGURE 5: CALCULATION OF THE PRICE AT WHICH AN ORDER IS WILLING TO TRADE

Trading in Locked Markets

Section 4.3.1 describes "Locked and Crossed Markets": "The UBS ATS will not effect a cross if the inside market for the stock is crossed (where the bid price exceeds the offer price), but will effect a cross if the market for a stock is locked (where the bid price is equal to the offer price); provided however, if instructed by an Order Originator, the UBS ATS will not execute a Pegged Order if the market for the stock is locked. In the event of an execution during a locked market, the cross will be executed at the locked price."

We first encode the classification of the current market data in IML:

```
type mkt_cond = MKT_NORMAL | MKT_CROSSED | MKT_LOCKED

let which_mkt (mkt_data) =
  if mkt_data.nbo = mkt_data.nbb then MKT_LOCKED
  else if mkt_data.nbo < mkt_data.nbb then MKT_CROSSED
  else MKT_NORMAL
```

FIGURE 6: DEFINITION OF MARKET COND TIONS

We then use the classification to determine (based on client settings) whether a pegged order may trade:

```
let good_mkt (o, mkt_data) =
  match which_mkt (mkt_data) with
  | MKT_CROSSED -> false
  | MKT_NORMAL -> true
  | MKT_LOCKED ->
    if (o.order_type = PEGGED) || (o.order_type = PEGGED_CI) then
      not (o.cross_rest.cr_no_locked_nbbo)
    else
      true
```

FIGURE 7: COND TIONING TRADING ON CURRENT MARKET

roving The Specification Is Compliant With Regulation

With our IML encoding of UBS ATS[5], we can turn to reasoning about whether the design of the venue is compliant with regulatory directives. We will later use results of this reasoning to construct high-coverage test suites for testing production systems. But, first we must ensure that our design is correct and compliant!

Sub-Penny Pricing

Our first verification goal concerns the requirement that a venue cannot accept orders priced off tick. This requirement is to ensure that no order can gain queue priority by providing economically insignificant price improvement. Page 5 of the Form ATS states this clearly: "Subpenny executions will not occur except at the mid-point unless the stock is trading below $1.00."

```
(* Sub_penny_price: return true if the price is sub-penny, unless it's also the market mid-point, and false
   otherwise *)
let sub_penny_price (p, tick_size, mkt_data) =
  let is_sub_penny = (ceil (p /. tick_size)) <> (floor (p /. tick_size)) in
  if is_sub_penny then
    not(price_eq (p, (mid_point (mkt_data))))
  else false

(* Check_orders: iterate through a single side of the book and returns true if sub-penny order exists *)
let rec check_orders (side, orders, tick_size, mkt_data) =
  match orders with
  | [] -> false
  | x::xs -> sub_penny_price (effective_price (side, x, mkt_data), tick_size, mkt_data))
             || check_orders (side, xs, tick_size, mkt_data)

(* Not_in_book: return true if there exist no orders that are sub-penny priced and not equal to mid market *)
let not_in_book (s) =
  let not_in_buys  = check_orders (BUY,  s.order_book.buys,  s.static_data.tick_size, s.mkt_data) in
  let not_in_sells = check_orders (SELL, s.order_book.sells, s.static_data.tick_size, s.mkt_data) in
  (not_in_buys && not_in_sells)

(* Sub_penny_pricing: no sequence of system events may result in an order with sub-penny effective price *)
(* s and s' represent arbitrary symbolic states of the venue. 'simulate' symbolically executes the venue *)
verify sub_penny_pricing (s, s') =
  (s' = simulate (s)) ==> not_in_book (s')
```

FIGURE 8: VERIFICATION GOAL FOR THE SUB-PENNY RULE

Figure   lists the corresponding Imandra verification goal. For presentation purposes, we elide the conditioning on $1.00 prices.

The key lines are the last two: they dictate that regardless of the venue's initial state, once its matching and communication logic processes all messages and trades all eligible orders, there will be no orders in the order book with sub-penny prices. This covers infinitely many possible combinations of orders sent to the venue and operator instructions to update any of the venue settings. (For more information on how Imandra is able to analyse infinite state spaces, please see our white papers "Creating Safe and Fair Markets" and "Transparent Order Pricing and Priority").

Is the encoding above the only way to define such a verification goal? Absolutely not. We leave the exact formulations of VGs to regulators and the industry to work out together. Our purpose is to create a scientifically based and rigorous *medium* for expressing and reasoning about financial algorithms.

[5] Disclaimer: the actual Form ATS is ambiguous for the reasons we discuss and hence our encoding may deviate from intentions of UBS. We have not consulted with the firm in the course of designing the model.

Once Imandra verifies this goal, it can then be used to generate a high-coverage test suite to run against the actual implementation of the model, i.e., the production system.

Crossing Constraints

Our second example highlights another issue raised by the SEC. Most dark pools, including the UBS ATS, implement rules restricting eligible (from the pricing perspective) orders from trading with each other. For example, such functionality can stem from the need to restrict self-crossing for fund managers that have to execute their trades on the market. That restriction is legal and expected, but there may be other restrictions that are not necessarily illegal, but may become so if they are not disclosed to all participants and/or the regulators. This is the second issue raised in the SEC case. The current Form ATS lists the current restrictions in Section 3.3 and we use these in our model.

The dark pool is a complicated trading engine with many inputs. How can we isolate a subset of these inputs and *mathematically* verify that they are the *only* factors that may prohibit two eligible orders from trading with each other? This is straightforward with Imandra's Information Flow Analysis.

Intuitively, here's how we will setup the verification goal:

- Imagine two possible scenarios (or "arbitrary states") of the venue, S_1 and S_2.
- Let us fix Buy_1, Buy_2 and Sell_1, Sell_2 to be the best bids and best offers, respectively, for the two scenarios.
- Further, let us state that scenarios S_1 and S_2 are indistinguishable with respect to the list of restrictions declared within Form ATS.
- Then, when we execute the model on those scenarios, they will either both result in fills or not execute. In other words, the outcome will be the same between those two scenarios.

If this statement is true for all possible configurations of the venue and other inputs into the system, then we know that those restrictions we isolated in the verification goal are the only restrictions that can prohibit execution of those orders. It's worth reiterating that there are different ways to encode such goals and this is just one of them.

```
verify isolate_cross_constraints (s_one, s_two, s_one_next, s_two_next, b1, b2, s1, s2, fill_1, fill_2) =
  (
   (* Set our best bid and offer *)
   Some b1 = best_buy (s_one) &&
   Some b2 = best_buy (s_two) &&
   Some s1 = best_sell (s_one) &&
   Some s2 = best_sell (s_two) &&

   (* Ensure that the crossing constraints match *)
   b1.cross_restrict = b2.cross_restrict &&
   s1.cross_restrict = s2.cross_restrict &&
   s_one.mkt_data = s_two.mkt_data &&

   (* Check that the orders can trade with each other *)
   prices_cross (b1, s1, s_one.mkt_data) &&
   prices_cross (b2, s2, s_two.mkt_data) &&

   (* The next states are generated by the model after it processes all messages and executes all eligible trades *)
   s_one_next = simulate (s_one) &&
   s_two_next = simulate (s_two) &&
 (* Identify corresponding fills in the next states *)
 fill_1 = get_fill_bounded (b1, s1, s_one_next.fill_log) &&
 fill_2 = get_fill_bounded (b2, s2, s_two_next.fill_log)
)

 (* Then the fills should match - either they're empty or the quantity and the price are exactly the same *)
==>  (fills_match (fill_1, fill_2))
```

FIGURE 9: VERIFICATION GOAL FOR CROSSING CONSTRAINTS

Transitivity of Order Ranking

Our original plan was to encode the two verification goals addressed in the SEC complaint, together with a family of goals related to regulatory properties of various order types. The latter is part of our standard offering to our clients for analysing their venue matching logic. As we were encoding the model, Imandra discovered subtle but fundamental issues in UBS's Form ATS description of its dark pool matching logic. We describe our findings in this section.

As already mentioned, *transitivity* is a basic requirement for 'stable' sorting operations. Simply put, it does not make sense to sort a list of objects (e.g., a list of orders in an order book) if the criteria by which you are sorting them is not transitive.

Recall the definition of transitivity: A relation (x R y) is transitive if and only if [(a R b) and (b R c)] always implies that [(a R c)]. If you imagine "R" as being ">" (greater-than), then it's easy to get an intuition for what transitivity means: [(a > b) and (b > c)] always implies that [(a > c)].

Consider now a function `order_higher_ranked` that computes whether or not one order should be ranked above another in the order book. If `order_higher_ranked` is not transitive, then you simply cannot use it to sort orders. If you did, then the priorities given to different kinds of orders would not be stable, and clients would not be able to anticipate matching behaviour. Such a flaw would be very difficult, if not impossible, to isolate by looking at the post-trade data alone.

Figure 11 has the corresponding IML code encoding the order ranking logic described in the Form ATS (subject to our understanding). The function `order_higher_ranked` takes the side indicator, order X, order Y, the structure with current NBBO and returns True if X takes priority over Y, False otherwise.

Once we submitted the code, Imandra replied within two seconds with an error: The order sorting function does not make sense, as the relation used to sort orders is not transitive. We then asked Imandra to explicitly compute for us a "counterexample," i.e., concrete inputs into `order_higher_ranked` that will cause it to violate transitivity:

```
verify rank_transitivity (side, order1, order2, order3, mkt_data) =
  (order_higher_ranked(side, order1, order2, mkt_data) &&
   order_higher_ranked(side, order2, order3, mkt_data) &&
==>
  (order_higher_ranked(side, order1, order3, mkt_data))
```

FIGURE 10: VERIFICATION GOAL FOR ORDER RANKING TRANSITIVIT

```
let order_higher_ranked (side, o1, o2, mkt_data) =

        let ot1 = o1.order_type in
        let ot2 = o2.order_type in

        let eff_price1 = effective_price (side, o1, mkt_data) in
        let eff_price2 = effective_price (side, o2, mkt_data) in

        let wins_price = if side = BUY then (if eff_price1 > eff_price2 then 1
                                             else if eff_price1 = eff_price2 then 0
                                                  else -1)
                         else (if eff_price2 < eff_price2 then 1
                               else if eff_price1 = eff_price2 then 0
                                    else -1) in
        let wins_time = if o1.time < o2.time then 1
                        else if o1.time = o2.time then 0
                             else -1 in

        let ci (ot) = (ot = PEGGED_CI || ot = LIMIT_CI ) in

        if wins_price = 1 then true
        else if wins_price = -1 then false
        else if ci (ot1) && ci (ot2) then
                   if o1.leaves_qty > o2.leaves_qty then true
                   else if o1.leaves_qty < o2.leaves_qty then false
                        else (wins_time = 1)
            else if wins_time = 1 then true
                 else if wins_time = -1 then false
                      else (match ci (ot1), ci (ot2) with
                              | false, false -> true
                              | false, true -> true
                              | true, false -> false)
```

FIGURE 11: ORDER RANKING FUNCTION

When Imandra was asked to prove the transitivity verification goal (Figure 10), it produced the following counter example:

```
order1 = {                     order2 = {                        order3 = {
       peg = NEAR;                    peg = NEAR;                       order_type = LIMIT_CI;
       order_type = PEGGED;           order_type = PEGGED_CI;           qty = 8400;
       qty = 1800;                    qty = 8400;                       price = 10.0;
       price = 10.5;                  price = 12.0;                     time = 236;
       time = 237;                    time = 237;                       ...
       ...                            ...                               };
       };                             };
```

FIGURE 12: COUNTEREXAMPLE TO ORDER RANKING TRANSITIVIT

Transitivity is violated because Order 1 takes priority over Order 2, and Order 2 takes priority over Order 3, but Order 1 DOES NOT take priority over Order 3! Why is this the case? Before we answer that question, it's important to note that all three orders have exactly the same effective price (the price at which they're willing to execute): 10.0. Note that the effective price is a function of the order type, peg level, limit price, NBBO, etc.

Here's the breakdown of why transitivity does not hold:

- Order 1 takes priority over Order 2 because: both orders share the same effective price and time, but Order 2 is a CI order. Therefore, Order 1 takes priority. Here's the culprit: "*For orders with the same price and time, priority is given to Resident and IOC Orders over Conditional Indications.*"
- Order 2 takes priority over Order 3 because: since they're both CI orders and share the same effective price, priority is then assigned based on quantity. Here's the exact quote: "*Invites are sent to the Order Originators of Conditional Indications on a priority based first on price, second on the quantity and third on the time of receipt by UBS ATS.*"
- Order 1 DOES NOT take priority over Order 3 because: Order 3 is older (timestamp = 236) than Order 1 (timestamp = 237).

Why is this so important? If a ranking function used to sort the orders is not transitive, then the priority logic is nonsensical and the results of "order sorting" cannot be trusted.

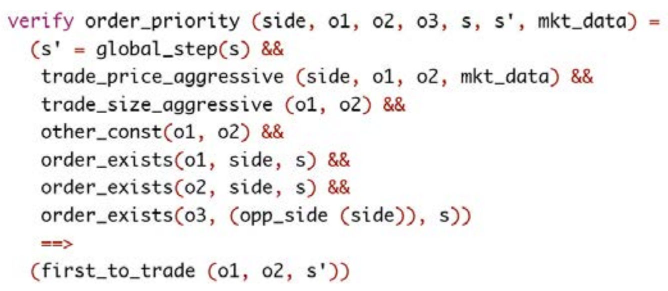
It's worth reiterating that we have no knowledge of the actual implementation of the UBS ATS. We base our analysis solely on the description given in Form ATS. But, if there is a discrepancy between the matching logic described in Form ATS and the actual implementation, then this is of course a major problem as well.

This example exemplifies why modern finance needs automated tools like Imandra that can reason about algorithms. The algorithms have become far too complex to manage by hand.

Order Priority Rules

Our last example demonstrates the application of Imandra to reasoning about order prioritisation rules. This example is motivated by numerous debates as to the merits of the abundance of different order types across the global markets. We argue that the complexity of modern market microstructure is not 'bad' in itself. The challenge, however, is to have the appropriate tools that allow market participants to analyse the offered order types, ensure they understand their benefits and that their systems are implemented to correctly interact with those venues.

Let us encode in IML a simple property: If the effective price of Order 1 is at least as aggressive as Order 2, and given that they have the same arrival time, have the same quantity and share crossing constraints, then Order 1 should trade first. This makes economic sense - if you're first and you're more aggressive than the rest, then you should always trade first (given that minimum quantity is met, you're not restricted, etc.). Here's how we would encode such property as a verification goal in Imandra:



```
verify order_priority (side, o1, o2, o3, s, s', mkt_data) =
  (s' = global_step(s) &&
   trade_price_aggressive (side, o1, o2, mkt_data) &&
   trade_size_aggressive (o1, o2) &&
   other_const(o1, o2) &&
   order_exists(o1, side, s) &&
   order_exists(o2, side, s) &&
   order_exists(o3, (opp_side (side)), s))
   ==>
  (first_to_trade (o1, o2, s'))
```

FIGURE 13: ORDER PRIORITY VERIFICATION GOAL

When we asked Imandra to verify this of the UBS ATS model, it came back with a counterexample. It

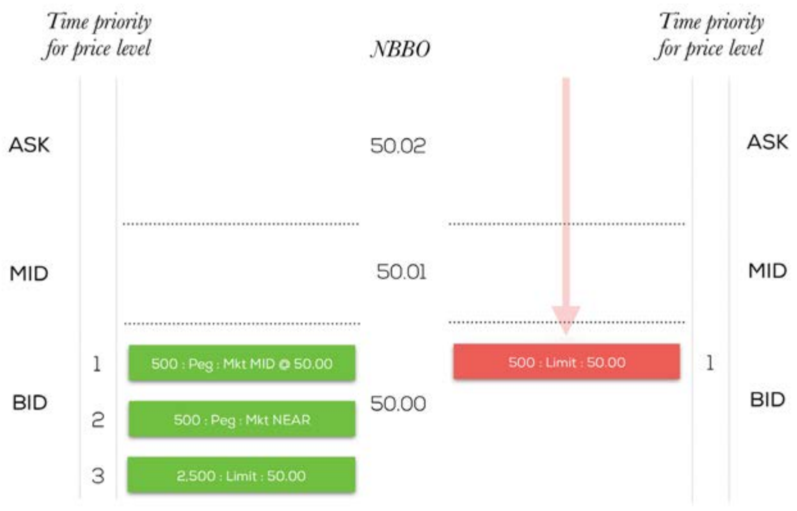


FIGURE 14: ORDER BOOK HAS AN INCOMING SELL ORDER

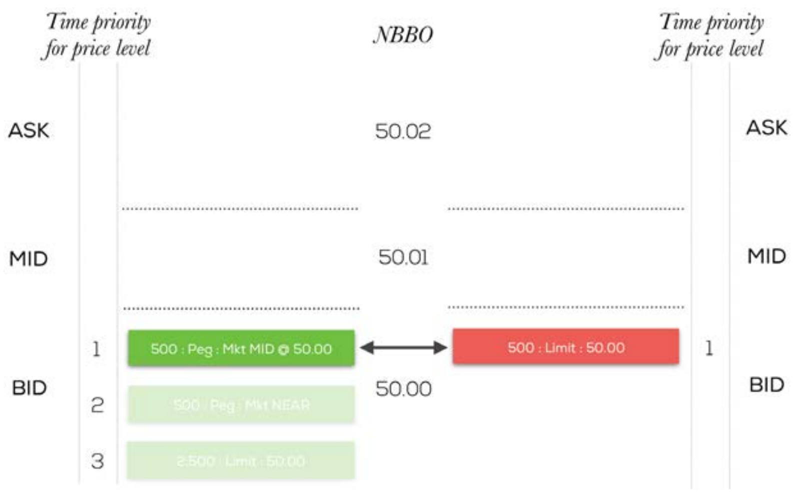


FIGURE 15: PegLimitConstraintMode = 1

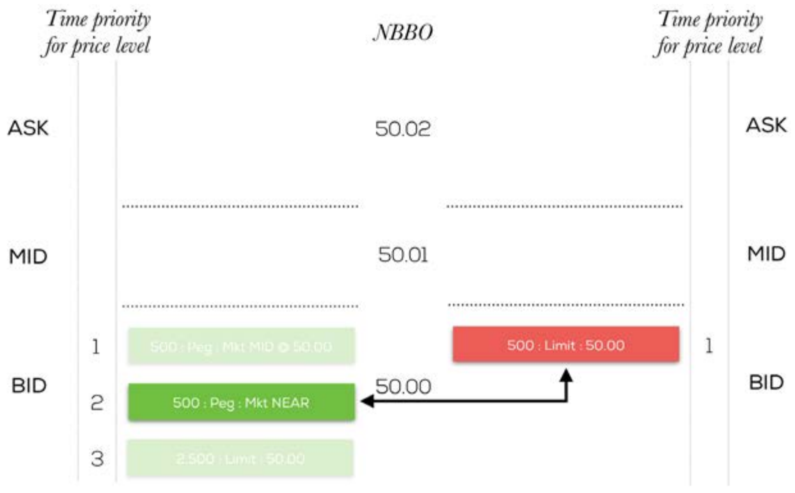
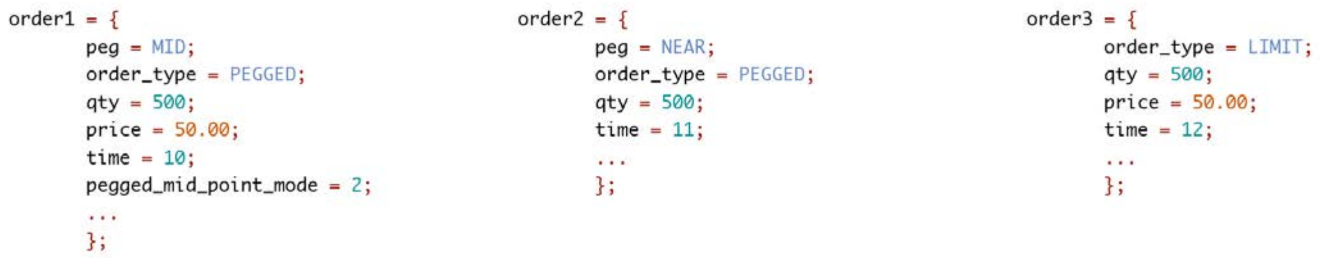


FIGURE 16: PegLimitConstraintMode = 2

turned out the VG failed because of a feature (if selected by the client) within the UBS ATS design that prevents an eligible pegged to MID order from trading if its limit price is less aggressive than the market MID. The setting that allows for this is called "PegLimitConstraintMode" (see, e.g., examples 5 and 6 of the UBS Form ATS). When clients request to set this value to 2, it will not trade. Alternatively, it will execute.



```
order1 = {
        peg = MID;
        order_type = PEGGED;
        qty = 500;
        price = 50.00;
        time = 10;
        pegged_mid_point_mode = 2;
        ...
        };
```

```
order2 = {
        peg = NEAR;
        order_type = PEGGED;
        qty = 500;
        time = 11;
        ...
        };
```

```
order3 = {
        order_type = LIMIT;
        qty = 500;
        price = 50.00;
        time = 12;
        ...
        };
```

FIGURE 17: COUNTEREXAMPLE TO ORDER PRIORITY VERIFICATION GOAL

Conclusion

With a focus on the UBS ATS and UBS's recent $14mm settlement with the SEC, we have demonstrated how Imandra radically improves the process of designing, implementing and regulating financial algorithms.

Our mission is to provide financial markets and regulators with powerful tools for managing the complex algorithms underlying modern trading systems and venues. Imandra by Aesthetic Integration brings revolutionary advances in formal verification to bear on financial algorithms, at last allowing us to scale robust engineering methods used in other safety-critical industries to finance.

We are driven by the fundamental improvements Imandra will bring to global financial markets. Significant portions of the costs and resources required to operate and regulate trading businesses will be eliminated. Precision and systematic rigour will replace ambiguous and ad hoc approaches to managing complicated trading systems.

Imandra will help you build safer, more stable and compliant businesses. Together let's make financial markets safe and fair.

About Aesthetic Integration

Aesthetic Integration Ltd. (AI) is a financial technology startup based in the City of London.

Created by leading innovators in software safety, trading system design and risk management, AI's patent-pending formal verification technology is revolutionising the safety, stability and transparency of global financial markets.

Imandra

- Brings major advances in formal verification to bear on trading systems and venues, delivering fully automatic analyses of your trading infrastructure
- Verifies correctness and stability of system designs for regulatory compliance
- Uncovers nontrivial bugs
- Creates high-coverage test-suites
- Radically reduces associated costs

As you design and implement trading systems and venues, Imandra's patent-pending technology helps you lay a stronger foundation for your future.

Legal Notice