AESTHETIC INTEGRATION

Aesthetic Integration Limited

1 Fore Street
London, EC2Y 9DT
United Kingdom

February 26th, 2016

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

Re: (**Release No. 34-76474; File No. S7-23-15**)

Dear Mr. Fields,

We appreciate the attention the Commission is placing on the contents and format for disclosing operational details of ATSs. We thank the Commission for this opportunity to comment on Reg ATS-N.

Given the staggering complexity of modern financial algorithms, the currently employed (ad hoc, prose-based) methods of disclosing client-facing (or counterparty-facing) algorithm specifications are no longer appropriate. The Commission must require financial firms to provide precise, mathematically analyzable specifications of their algorithms. Thankfully, there is a mature and effective field of science devoted to the precise disclosure, analysis, and regulation of complex algorithms: the field of *formal verification*. Other safety-critical industries like avionics and hardware design already rely on formal verification to make their algorithms safe. By requiring financial firms disclose their algorithms in precise mathematical terms, the Commission can bring deep advances in the scientific analysis of algorithms to bear on market safety, stability and transparency.

We enclose our official commentary along with two recent white papers presenting concrete applications of our proposal.

About Aesthetic Integration

Aesthetic Integration Ltd. (AI) is a financial technology start-up bringing cutting edge formal verification technology to financial markets. AI is working with leading financial institutions to revolutionize the process of designing, implementing and ensuring compliance of complex trading systems. In late 2015, AI won the UBS Future of Finance Challenge, coming in 1st out of 620 companies from 52 countries. AI also won a Futures Industry Association Innovator Award at the FIA Expo 2015 in Chicago.

Denis Ignatovich, co-founder of AI, has over a decade of experience in trading, risk management, quantitative modeling and complex trading system design. Prior to joining AI, he was head of the central risk

trading desk at Deutsche Bank London. He holds an MSc in Finance from the London School of Economics and undergraduate degrees in Computer Sciences and Finance from the University of Texas at Austin.

Dr. Grant Passmore, co-founder of AI, has more that ten years' industrial formal verification experience. He has been a key contributor to safety verification of algorithms at Cambridge, Carnegie Mellon, Edinburgh, Microsoft Research and SRI. He earned his PhD in Automated Theorem Proving from the University of Edinburgh and is a Life Member of Clare Hall, University of Cambridge.

Overview

We share and shall discuss the Commission's concern for the lack of dark pool transparency and related issues of best execution. Furthermore, the current ad hoc and opaque methods of communicating client-facing algorithm specifications have many indirect negative effects:

- A lack of operational transparency increases costs for brokers connecting to venues, and they in turn pass on these costs to clients.

- Opaque disclosures increase operational risks for brokers trading on the venues, hence increasing the likelihood of 'glitches' occurring.

- These increased risks compound in a fragmented national market system, with a multitude of opaque venues competing with each other. The resulting instability poses a nontrivial threat to national security[1].

By not requiring that dark pool / Form ATS-N disclosures contain a precise specification of the venue matching logic, the Commission places itself at a significant disadvantage, effectively relegating its regulatory powers to the analysis of post-trade data. If instead, dark pool operators were required to provide precise mathematical specifications of their matching logics, the Commission could rigorously analyze their designs (and marketing materials) in advance, and with far more powerful tools.

We would like to bring the Commission's attention to recent breakthroughs in formal verification that allow us to apply similar techniques to those used in avionics and hardware manufacturing to regulate complex financial algorithms. We see these breakthroughs as fundamental game-changers for the Commission's mandate.

As a useful analogy, consider how a breathalyser is used to determine whether a person is intoxicated. Without such a tool, the burden on authorities (and those suspected of intoxication) is significant: much circumstantial evidence must be gathered, and ad hoc, unscientific tests conducted. But, by transforming the problem of determining intoxication into the domain of chemistry, the situation is markedly improved.

Similarly, if the problem of dark pool compliance is transformed into the appropriate scientific domain, in this case into the realm of formal verification, then we can significantly improve safety and fairness of financial markets while simultaneously cutting costs for the Commission and the regulated institutions.

---

[1] "'Flash crash' trader to appear in court to fight extradition to US" is available from www.theguardian.com

The Precise Specification Standard

We are encouraged by your statement that the data submitted by venue operators should be in 'machine-readable format,' in contrast with the existing method of publishing (when they're made publicly available) scanned PDFs that are not easily accessible by basic word search functionality. We argue the Commission should go further in requiring the operational details submitted be analyzable by modern automated reasoning techniques. Such formatting would allow the Commission to tap into the field of formal verification, in a manner similar to how the FAA and DoD analyze and regulate complex, safety-critical algorithms.

We see the following issues when an operator submits an English Description (ED) of the operational details of a venue:

1. An ED is completely detached from the actual production system. So, if a member of the development team makes a subtle, yet significant change to the logic of the system (perhaps accidental), then the 'break' with the ED is very difficult to detect; checks for this are not automated.

2. An ED is typically describing a system that may be in a virtually infinite number of possible configurations. In practice, the possible behaviors of such a system cannot be exhaustively analyzed "by hand," i.e., by a team of regulators reading and attempting to understand the ED.[2]

3. An ED cannot be used to test implementations of systems that are trading on the venue described. In other words, the ED cannot be incorporated into the development process of trading systems that submit orders to the ATS.

In order to remedy the issues above, we suggest the following core requirements on any regulatory format for disclosing operational details of an ATS. We call a disclosure format meeting these requirements a Precise Specification (PS).

1. In a PS, the operational details of an ATS should be described in an executable programming language with a formal semantics. We give detailed examples of this in our attached white papers. The phrase 'formal semantics' refers to the ability to translate the algorithm disclosed into a precise mathematical model. This model can be analyzed using formal verification techniques to automatically check its logic for potential violations of many key regulatory directives.

2. Many of the Commission's requests for comments ask whether certain information pertaining to the actual operation of a venue should be disclosed. We propose a simple test that should help the venue operators and the Commission answer that question:

   Disclosure Test: *Is the information necessary for one to write an observationally-equivalent simulator of the venue? If so, it should be required.*

   In a PS, the Disclosure Test must be passed. One byproduct of this is that, given a PS, one can directly create an executable "simulator" of the venue design that was disclosed. Different levels of observational-equivalence can be utilized for different contexts and purposes. As a base-line, we suggest the

[2] Please see our white paper "Case Study: 2015 SEC Fine Against UBS ATS" for a concrete example.

observational-equivalence of two venue implementations require equality of message sequences (FIX, ITCH, etc.) over `replays.'

An operator should judge the disclosed PS complete if and only if it is reconcilable with actual post-trade data of the production system. That is, the specification given should be convertible into a machine-executable program that can be run against actual historical data. In doing so, its faithfulness to the behavior of the production system may either be confirmed or refuted over a given time window.

Given a PS, the Commission and financial firms can leverage formal verification techniques to:

1. Automatically analyze the submitted specifications for potential violations of many key regulatory directives. For example:
   - Does the ATS accept sub-penny orders?
   - Is the order ranking criteria transitive?
   - Is there any subtle combination of order parameters that allow someone to 'jump the queue'?

2. Allow those trading on the venues to automatically test their connectivity and verify their routing algorithms (for best execution principles) using quantitative state-space coverage metrics.

For more specific examples we urge the Commission to review our recent white papers detailing an application of our formal verification system, Imandra, to analyzing safety and fairness properties of venues. In fact, our latest white paper "Case Study: 2015 SEC fine against UBS ATS" follows in detail the Commission's order against UBS ATS. The case study formed part of our application into UBS's Future of Finance Challenge that was held in (August - December) 2015. Aesthetic Integration was selected as the first place winner out of more than 600 applicants from 52 countries. In the case study, we detail how issues raised in the SEC settlement, namely sub-penny pricing and undisclosed trading constraints, may be encoded as mathematical properties and automatically analyzed for a venue specification.

About Formal Verification

We published a white paper in 2015, 'Creating Safe and Fair Markets'[3], 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[4] precise levels of system testing and formal verification within both the Common Criteria Evaluation Assurance Levels and DO-178C[5] frameworks. Safety-critical algorithms such as air 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.

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

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

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

- The Department of Transportation has commissioned work[6] 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[7] framework relies on formal verification to ensure its safety.

- The Department of Defense[8] leverages formal verification across numerous applications, including the design and regulation of cryptographic algorithms and secure hypervisors.

Client Connectivity

Consider the efforts required of a designer/developer of a Smart Order Router that is operating in the US markets. That person must ensure that their system: 1) understands the precise descriptions of all of the venues on which it trades, 2) is safe (e.g., will not submit incorrect limit prices[9]), and 3) is compliant (and be able to demonstrate this).

Key aspects of these tasks may also be transformed into the realm of formal verification. Our next public commentary in response to CFTC's Reg AT proposal will address these issue in more detail.

Concluding Remarks

The algorithms underlying modern financial markets have become too complex for the traditional tools and methods used to design and regulate them. With recent breakthroughs in formal verification, we can bring modern scientific rigor to the design and regulation of financial algorithms.

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.

Again, we thank the Commission for the opportunity to voice our comments on the proposal.

Sincerely,

Denis Ignatovich
Co-Founder, AI

Grant Passmore, PhD
Co-Founder, AI

---

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

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

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

[9] "17-minute trading glitch put Goldman's reputation on the line" available from http://on.ft.com/1dwK5jq

AESTHETIC AI INTEGRATION

TRANSPARENT ORDER PRIORITY AND PRICING

Transparent Order Priority and Pricing

Denis A. Ignatovich and Grant O. Passmore
Aesthetic Integration Limited

Abstract

Transparency of trading algorithms is a pressing issue in today's financial markets. For example, though a dark pool may have access to client-specific information, can it ever use that information to affect order priority or pricing in a manner not reflected in regulatory filings and marketing materials? Many such questions are difficult (if not impossible) to answer by looking at post-trade data alone. Fortunately, recent scientific breakthroughs allow us to systematically analyse algorithms for compliance and conformance with regulatory directives and marketing materials.

Our product, Imandra, leverages latest research breakthroughs to deliver an automated formal verification solution for designing, implementing and regulating financial algorithms. Imandra reasons about precise statements concerning the behaviour of trading algorithms, giving the industry and regulators unprecedented insight into what trading algorithms can and cannot do.

In this report, we illustrate Imandra's application to the transparency of order priority and pricing within venues.

Modern trading systems are highly nontrivial engineering artefacts processing tremendous volumes of data at lightning speed. These algorithms must operate in a dynamic environment, adapt to ever-changing client demands, and abide by numerous regulatory and internal controls. Despite this complexity, venue operators must demonstrate to their clients and regulators that the underlying algorithms are compliant with numerous regulatory directives, and that they in fact perform as described in marketing materials.

Human reviewing of the actual source code of these algorithms is infeasible: First, there is simply too much of it, and second, it is in a constant state of flux. An alternative approach is the analysis of post-trade data. Unfortunately, this data is typically noisy to a degree that hampers reliable analysis. Moreover, many important classes of design and implementation flaws are impossible to identify from post-trade data alone, even if the data were perfectly intact. Another shortcoming is that a post-trade approach is retroactive - it does not provide a preemptive solution for ensuring future designs and implementations are failure-proof. The current format for disclosing financial algorithms makes the job even more difficult and error-prone: A typical regulatory filing or marketing material is presented in

English prose in a manner unfit for automated processing and analysis.

We argue that questions of transparency of financial algorithms are symptoms of a fundamental problem: The complexity of financial algorithms has significantly outpaced the power of traditional tools used to design, implement and regulate them. The solution lies in the application of modern scientific methods developed precisely for analysing the behaviour of complex algorithms. Such methods have already become the backbone of engineering processes in other safety-critical industries like avionics and hardware manufacturing. Most recently, they have proved themselves indispensable in the design of safe collision avoidance kernels within self-driving cars.

In the following report, we demonstrate how our product Imandra can be used to analyse the design and implementation of venue matching algorithms.

Complexity

We've made several references to the term *complexity* when referring to modern trading systems. What *precisely* do we mean? What is the *source* of such complexity? And what *scientific* tools can we leverage to manage it?

One way to measure the complexity of a software system is to analyse its *state space*, a mathematical description of its possible behaviours. A *state* is a possible configuration of the system, i.e., the collection of all data contained in its memory at any given snapshot in time. For example, if you examine an order book on an exchange and see limit orders, that snapshot of the data contained within the venue represents a state. If you send an order to an exchange and part of it crosses while residual rests on the order book, that is a new state (or a sequence of new states). The matching rules, e.g., definitions of order types and triggers for transitions into volatility auctions, define how the venue *transitions* between states. To provide some intuition: How many possible distinct sequences of orders can be sent to a dark pool by all of the firms that trade within it? The answer is infinitely (or virtually infinitely) many. The structure of a financial algorithm's state space can be incredibly complex. We are long past the days when financial algorithm correctness can be ensured by hand.

Finance is not alone in having to deal with such complexity. 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 EASA require the use of rigorous mathematically-based methods for demonstrating safety of autopilot systems before they're allowed to be deployed.

So why hasn't finance adopted these techniques before today? Although the issues of managing complex algorithms in finance and, e.g., avionics, share much in common, the nature of algorithms in avionics is very different from those used in trading. Thus, we could not simply replicate the techniques used in those other industries; new techniques were needed.

Recent advances in formal verification (including Satisfiability Modulo Theories (SMT), automated induction and nonlinear decision procedures) allow us to scale automated reasoning techniques to the algorithms underlying trading systems. We've leveraged these results to create Imandra, a highly automated formal verification solution that delivers the power of tools similar to, e.g., those used by NASA engineers in designing safe autopilot systems, into the hands of trading professionals, while requiring no knowledge of the obtuse mathematics involved.

Imandra

Imandra allows one to ask deep questions about an algorithm's possible behaviours, and to analyse production implementations for conformance to their designs. Such questions are encoded as *verification goals* (VGs). To reason about VGs, Imandra employs powerful proprietary automated reasoning technology to decompose the infinite state space of the system logic to either: (i) find a concrete counterexample showing where the VG fails, or (ii) prove that it holds for all possible states of the system. If the VG holds of the system design, Imandra uses its symbolic state space decomposition to automatically construct test suites meeting rigorous quantitative test coverage metrics. These test suites may then be used to test production systems for conformance to verified designs.

The Imandra Modeling Language (IML) is used to encode trading system specifications and verification goals. IML is both a programming language[1] and a mathematical logic. In addition to being compiled and run, every program written in IML is automatically translated by Imandra into mathematics. Imandra's reasoning engine may then be used to analyse possible behaviors of the encoded algorithms.

To apply Imandra to reason about venues, we need the following:

- A specification of the matching logic (e.g., as described in Form ATS or exchange bylaws) expressed in IML. In addition to order type definitions, this specification contains definitions of the various parameters and attributes an order may have, the precise messaging format (e.g., FIX), and other details required to create a fully functional simulator of the venue.
- A verification goal expressed in IML. For example: *Does the venue accept 'sub-penny' orders*? In this report, we will demonstrate an application of Imandra to two VGs related to order priority and pricing.

Once Imandra is asked to verify that a VG holds of the design, it will convert the trading system design and VG into mathematical logic. There are many intricate steps that must take place for the conversion, but those details are hidden from the user.

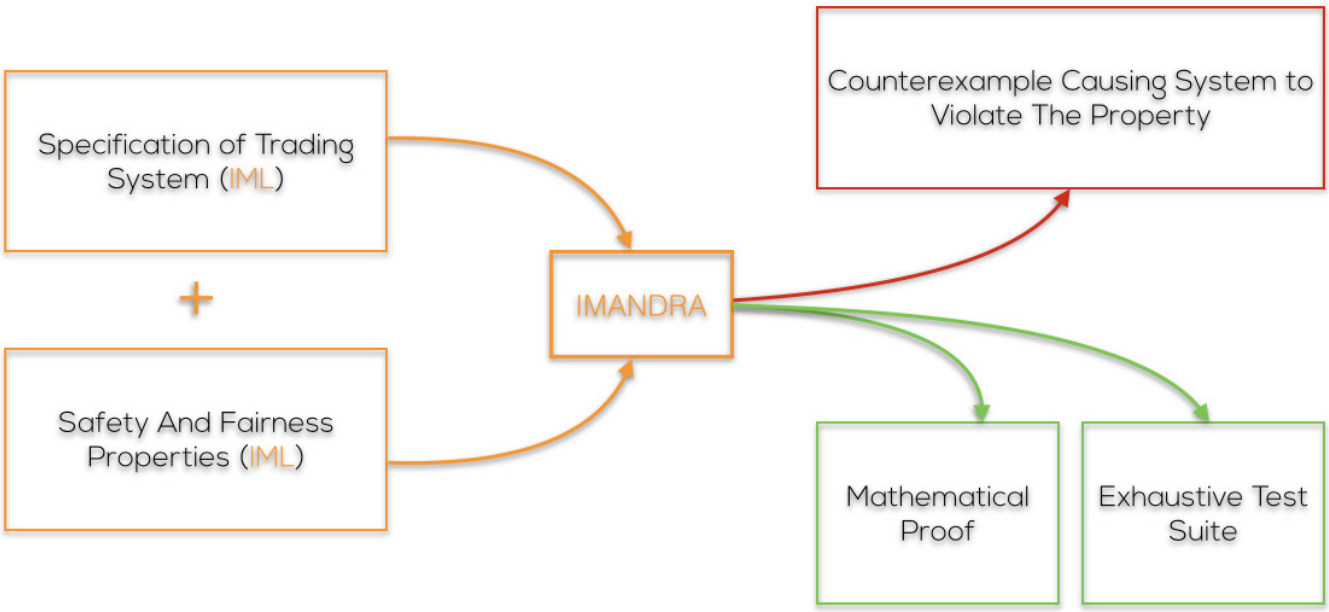

Figure 1: Overview of Imandra.

[1] IML's executable core is a subset of the OCaml programming language. Aesthetic Integration is a proud member of the Caml Consortium.

Imandra next uses its powerful automated theorem proving engine to create a logical representation of the state space and analyse the venue design for possible breaches of the verification goal in question. If there exists a state where the property is violated, Imandra will work to find it and convert it into a *counterexample,* a sequence of inputs into the system leading it to violate the VG. Alternatively, if the VG is true, then Imandra will work to produce a *mathematical proof* that the venue design cannot violate the property. Such proofs can be exported, e.g., to regulators, and be independently audited.

A verification goal can be thought of as a statement about a program that is either `true` or `false`. Formally, a VG is simply a function whose output is a boolean value. When we "prove a VG," we actually prove that it will evaluate to `true` *for all possible inputs.*

For example, consider the function definition '`let simple (x) = x > 5.`' We can execute this function, e.g., calling it with value `6` (i.e. '`simple (6)`', which evaluates to `true`). This is nothing new; it's just programming. Now, what is special about Imandra is that in addition to writing and executing programs, you may also write VGs and automatically reason about their behaviour:

```
verify simple_VG (x) =
  (x > 10) ==> (simple(x))
```

This VG states that if the input `x` is greater than `10`, then `simple(x)` will *always* be `true`. To prove `simple_VG`, Imandra analyses the definition of the function `simple`, turns it into mathematics, and reasons about it symbolically. This example is trivial, but the same principles apply to more complicated programs, e.g., as we include loops, recursive functions, complex data types and nonlinear calculations over many thousands of lines of code. In cases where an exchange has numerous order types, various trading phases, auctions, volatility circuit breakers, etc., Imandra's tremendous automated reasoning power becomes apparent.

A natural question to ask is: "*How do you know that Imandra is correct?*" If Imandra produces a counterexample to the VG in question, then this is easy to analyse for the user: The trading system can be executed upon the counterexample, directly illustrating the issue Imandra found. But what if Imandra produces a proof that a VG is *always* true, i.e., that no counterexamples to the goal exist? When reasoning about software, proofs can be very difficult to construct; Imandra fully automates this process for many key classes of algorithm properties[2]. But once found, they are easy to verify. In fact, there are open source third-party tools that can be used to certify Imandra's proofs.

Creating Precise Venue Specifications

A significant challenge for regulators (and those trading on exchanges) is that the documentation and marketing material given to them is often imprecise. It is commonly expressed in English prose, an obviously deficient way to communicate complicated mathematical objects. English descriptions of algorithms lead to ambiguity and open up opportunities for 'liberal' interpretations. But most importantly, English prose makes it impossible to automatically analyse algorithm behaviour for compliance with regulatory directives, or to reconcile such descriptions with production systems. The lack of precisely described rules by which venues operate also obstructs their clients from thoroughly testing their systems' connectivity to those venues, making the process tedious and expensive.

Let us now turn to some examples of Imandra in action. The two examples given below are based on Imandra models of a dark pool and an exchange. The model for the dark pool is based on several publicly available Form ATS filings. The exchange model is derived from the SIX Swiss Exchange's publicly available trading guide.

[2] There are problems, of course, that Imandra cannot solve. For example, if you ask Imandra to prove Fermat's Last Theorem, it will first attempt to do so, but will then come back to say 'Sorry, I give up.'

To provide some intuition about the models, here's a fragment of IML that declares order types that may be supported in a dark pool:

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

Once defined, the derived[3] venue simulator will automatically reject any orders sent to it (e.g., via FIX) that are not of those types. Furthermore, the structure of IML will force you to attach precise meaning to each declared order type.

In another example, the following fragment of IML is part of a calculation of the most aggressive price at which an order is willing to trade:

```
match o.order_type with
| LIMIT -> if side = BUY then
              if gte (o.price, mkt_data.nbo) then mkt_data.nbo else o.price
           else
              if lte (o.price, mkt_data.nbb) then mkt_data.nbb else o.price
| MARKET -> if side = BUY then mkt_data.nbo else mkt_data.nbb
```

Venues, whether dark pools or exchanges, share much in common with each other. Imandra has libraries of 'generic' models containing common venue components and other boilerplate code. These libraries allow our clients to focus on encoding components that are specific to their venues, significantly reducing the time required to implement a fully functional IML model.

Order Priority

Many recent regulatory directives contain behavioural constraints on trading algorithms. We will demonstrate how such directives may be converted into precise verification goals in IML.

Our first example demonstrates a verification goal encoding the constraint that no order type (along with some combination of its attributes) may 'jump the queue' under certain market conditions and operator settings.

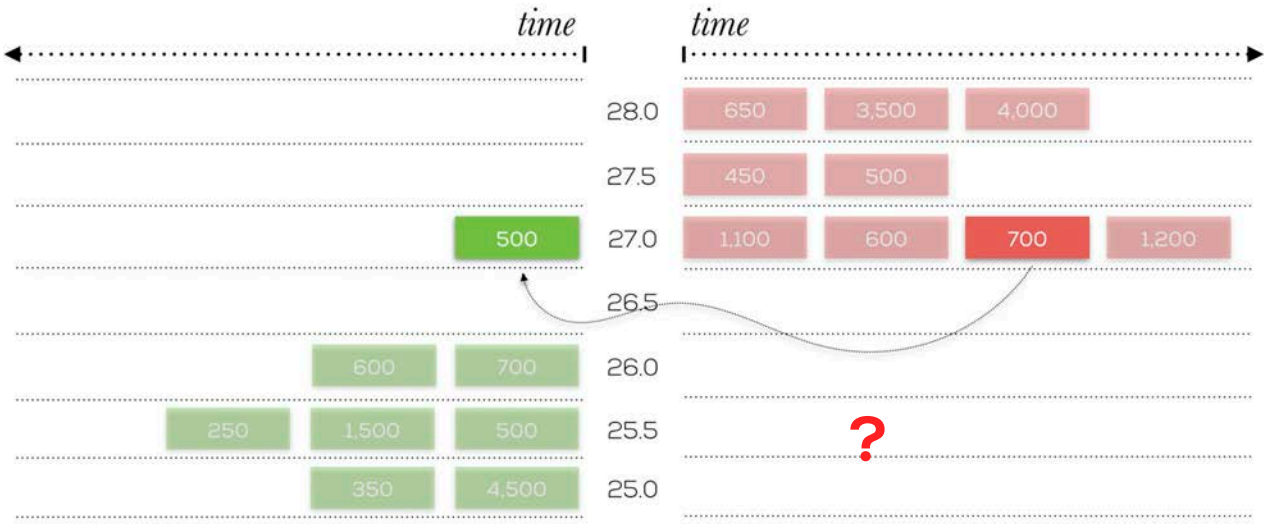

Figure 2: Out of priority, order to sell 700 matches against incoming market buy order for 500.

[3] IML venue models can be automatically compiled into high-performance venue simulators.

What are the typical factors determining an order's priority?

- *Price* - the price at which an order is willing to trade, which depends on the limit price, order type (e.g. Market or Limit) and the current National Best Bid and Offer (NBBO),
- *Time* - the time the order arrived,
- *Category* - client IDs may be categorised into groups with differing priority levels.

In addition to price constraints, there are many other constraints that may prevent two orders from trading with each other:

- *Minimum quantity than an order must trade,*
- *Self-crossing (for an individual client),*
- *No trading during specific market conditions (e.g., locked market),*
- *Round lot trades, … .*

There are many other 'areas' of the model that may affect how an order is prioritised and traded (e.g., the logic within a market data handler). Yet, we want to be able to encode a high-level statement about the venue and reason about it. For example: *If I send an order to the venue and my order is the most aggressive and it's been there longest, I should get filled before anyone else (unless there are restrictions that prevent a trade that are disclosed to me in the marketing materials).*

How might we express this statement as an IML verification goal? We'll start by saying there are orders 1 and 2 (on the same side), but order 1 is more aggressive and has an older time stamp than order 2. Furthermore, they have exactly the same restrictions (minimum quantity, crossing constraints, etc.). By *aggressive*, we refer to the price at which an order is willing to trade, which depends on order type and side. In this context, if an order arrives on the opposite side and it meets the prices of both orders 1 and 2, and is otherwise eligible to trade against both of them, then we would like to verify that order 1 will trade first. Here's the verification goal:

```
verify order_priority (side, o1, o2, o3, s, s', mkt_data) =
  (s' = next_state(s) &&
   order_at_least_as_aggressive (side, o1, o2, mkt_data) &&
   order_is_older(o1, o2) &&
   constraints_equal(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'))
```

This example underscores the power of Imandra: *Imandra allows you to encode high-level properties of the behaviour of an algorithm and reason about them automatically*. Such a verification goal is applied to the model of an entire venue, including the various risk gates and market data handlers. It is a universal statement that may be applied to almost any venue, regardless of the order types it supports or the specifics of its client categories. 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.

What can lead to a violation of this VG? Many (intentional or unintentional) design and implementation details. Such flaws may range from intentional decisions to prioritise internal clients, to accidental 'bugs' within the code failing to execute a trade during particular market conditions. Our next example showcases the often surprising results of applying formal verification techniques to non-trivial systems.

Order Pricing

Imandra's Information Flow Analysis allows you to analyse and isolate the effects of certain inputs into a trading system. In our next example, we use Imandra to analyse how client IDs attached to orders can affect prices of fills in an exchange model based on the public trading guide of the SIX Swiss Exchange.

We begin by stating our verification goal in 'plain English': *Client ID should not play a role in calculating the price of a fill.* It sounds straightforward, but considering the complexity of the system and the numerous decisions that affect whether an order is actually executed, it may be difficult to express and verify.

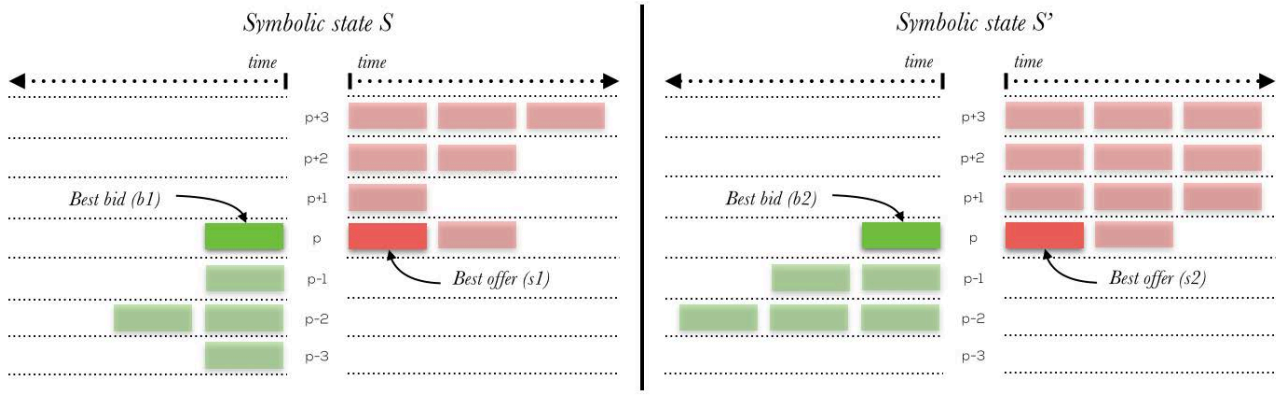

Figure 3: Our initial attempt to setup the verification goal.

Here is one way to setup this VG: Imagine there are two symbolic states[4] *S* and *S'* of the exchange with the following correspondence:

- The best bids in both states are exactly the same (i.e. quantity, limit price, order type, etc.) **except** for their client IDs.
- Similarly, the best offers in both states are exactly the same **except** for their client IDs.
- Furthermore, S and S' are equivalent in all state variables **except** for their order books.

Then, the pricing of a fill for S and S' should be identical.

[4] Recall that there exists a virtually infinite number of possible states. By a *symbolic state* we mean "a fixed but arbitrary state," just as you might use the symbolic variable 'x' to mean "a fixed but arbitrary real number" when doing an algebraic calculation.

Here is the corresponding verification goal:

```
verify match_price_ignores_order_source (s, s', b1, s1, b2, s2) =
     (orders_same_except_src (b1, b2) &&
      orders_same_except_src (s1, s2) &&
      states_same_except_order_book (s, s') &&
      best_buy s = Some b1 &&
      best_sell s = Some s1 &&
      best_buy s' = Some b2 &&
      best_sell s' = Some s2)
     ==>
     (match_price s = match_price s')
```

Naturally, we would expect the exchange to fill both pairs of orders at exactly the same price. This expectation reflects the venue's lack of preference for any specific client in the course of assigning a fill price when a trade is executed. However, when we ask Imandra to prove this goal, it returns in seconds with a counterexample (we are not displaying the entire counterexample, but rather only information relevant to our example):

```
State (S)
Buys:
     Time: 1, Type: Market, Attr: Normal, Src: client(13, G_MM, nil), Qty: 2
     Time: 38, Type: Market, Attr: Normal, Src: client(23, G_MM, nil), Qty: 25
Sells:
     Time: 449, Type: Market, Attr: Normal, Src: client(18, G_MM, nil), Qty: 2
     Time: 2437, Type: Limit, Attr: Normal, Src: client(29, G_MM, nil), Qty: 31, Price: 80.74

State (S')
Buys:
     Time: 1, Type: Market, Attr: Normal, Src: client(8, G_MM, nil), Qty: 2
     Time: 1796, Type: Market, Attr: Normal, Src: client(35, G_MM, nil), Qty: 37
Sells:
     Time: 449, Type: Market, Attr: Normal, Src: client(3, G_MM, nil), Qty: 2
     Time: 609, Type: Market, Attr: Normal, Src: client(42, G_MM, nil), Qty: 44
```

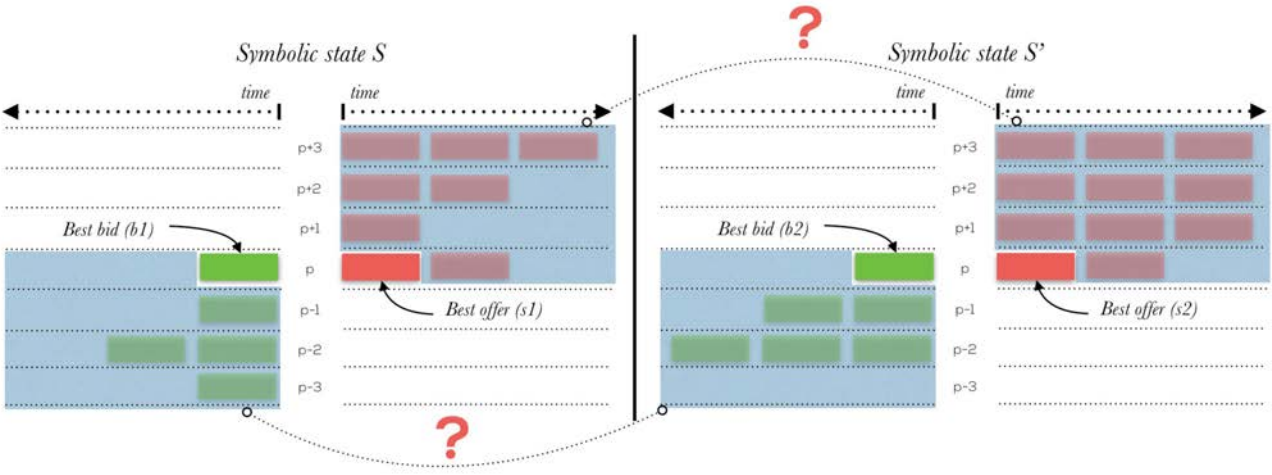

Figure 4: The counterexample pointed out a flaw in how we formulated our verification goal.

The counterexample illustrates a scenario where the verification goal is false! At closer inspection of the counterexample, we realise that our original formulation of the VG was naive. We failed to consider the case when best bids and offers in both states are Market orders. In such scenarios, the exchange transitions into an auction and other orders in the book can influence the price of the uncross.

We now extend our verification goal to take into account the equivalence of orders after the best bid and offer for each of the order books. Notice that we use the 'tail' function to refer to all orders in the book except for the very first one. If you recall, IML is both a programming language and a logic. So when we compile the IML model to use as a simulator, the 'tail' function will act 'normally': for example, when called with a list, [1; 2; 3], it will return [2; 3]. But, when we ask Imandra to reason about the possible behaviours of the IML model, 'tail' refers to all possible orders that may be in the order book *after* the best bid or offer. This value might be [] (i.e., there are no orders after the top order) or a list of 1,000,000 orders that have been sent to the venue by multiple clients. The key insight is that by using 'tail' symbolically, one is covering all cases.

Here is the updated VG with the constraint on the 'tails' of the order books:

```
verify match_price_ignores_order_source (s, s', b1, s1, b2, s2) =
    (orders_same_except_src (b1, b2) &&
     orders_same_except_src (s1, s2) &&
     states_same_except_order_book (s, s') &&
     List.tl s.order_book.buys = List.tl s'.order_book.buys &&
     List.tl s.order_book.sells = List.tl s'.order_book.sells &&
     best_buy s = Some b1 &&
     best_sell s = Some s1 &&
     best_buy s' = Some b1 &&
     best_sell s' = Some s2)
    ==>
    (match_price s = match_price s')
```

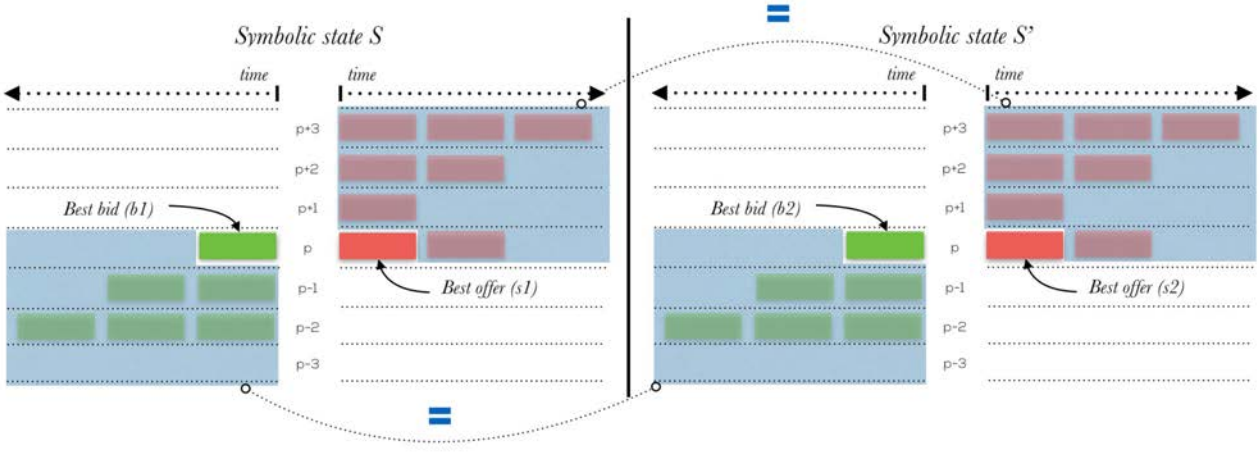

Figure 5: We amend our verification goal by placing additional constraint on 'tails' of the order books.

Now if we run Imandra on the updated VG, we get the following:

```
thm match_price_ignores_order_source = <proved>
```

Imandra successfully proves that our specification of the SIX Swiss Exchange matching logic is consistent with the verification goal we encoded. As a next step, we can ask Imandra for a trace of its reasoning, and can even request its proof be exported as formal evidence of compliance from Imandra.

Connecting with Production

Once you verify that the IML model is correct with respect to a verification goal (i.e., Imandra proves the VG), you can start to reason about whether a production system is faithful to its verified design.

Imandra contains a proprietary method for generating high-coverage test suites to analyse conformance of the actual implementation with the IML specification. Because Imandra analyses the infinite state space of the IML specification, it is able to discover important hard to find 'corner' cases that must be tested.

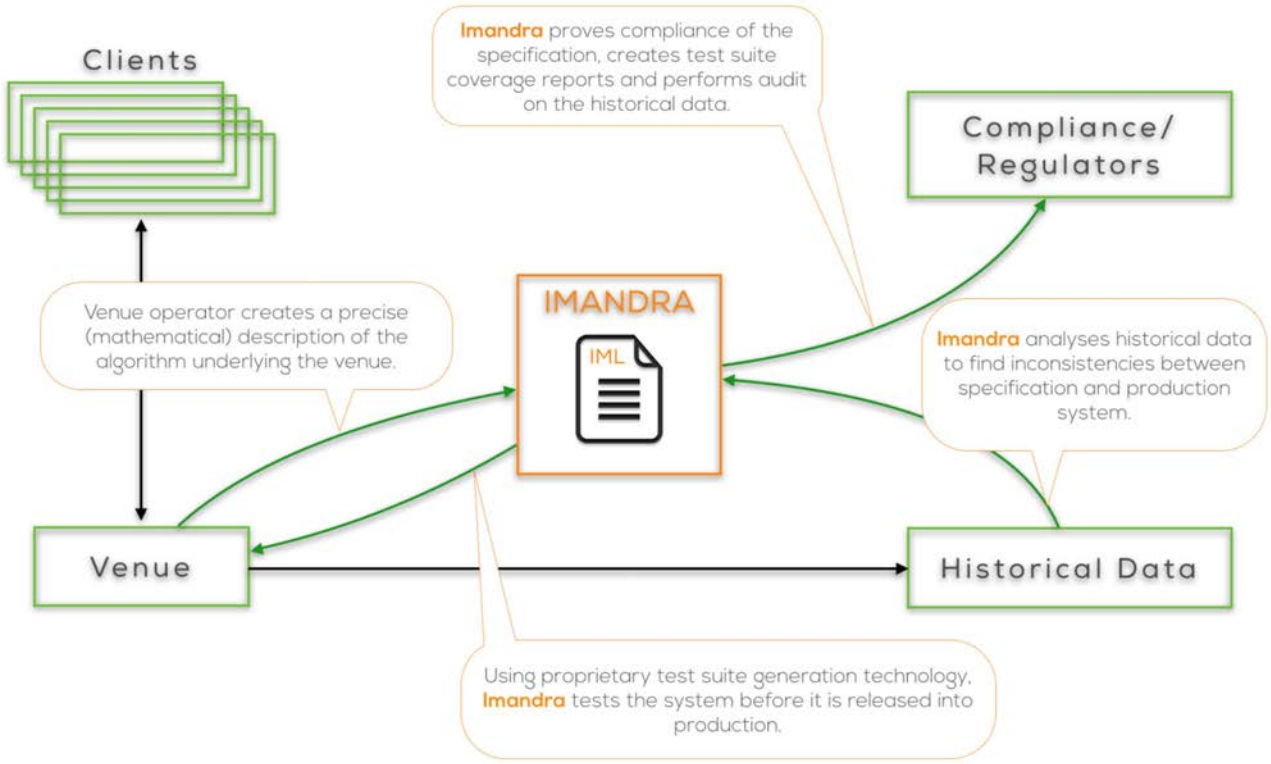

Figure 6: Integrating Imandra into design/development process.

Beyond Order Priority and Pricing Rules

Venues must operate under numerous complex constraints dictated by internal controls, customer demands and regulatory requirements. The highly intertwined matching logic of a venue makes it difficult to ensure that one component of a trading system does not 'overrule' another component resulting in unintended behaviours of the system. Imandra allows you to automatically reason about such entangled functionality to ensure system integrity.

We have demonstrated how Imandra can be used to reason about properties of venue order priority and order pricing. With Imandra, regulatory directives for financial algorithms can be encoded as precise mathematical statements, and Imandra's powerful automated formal verification techniques can be applied to analyse trading system designs and implementations. This profound increase in resource efficacy and rigour benefits both the industry and regulators.

Finally, let us end with a few more examples of venue verification goals that may be reasoned about with Imandra:

- *Pricing: does the venue allow sub-penny pricing?*
- *Reporting: are the trades tagged correctly and are they stored according to appropriate encryption requirements (i.e. is the client ID stored as raw text within the database)?*
- *Round-lot trades: does the venue abide by round-lot trading client restriction?*
- *Primary exchange: does the venue suspend trading when the primary is suspended?*
- *Limit Up/Down: will venue trade if the price is outside the LU/D bounds?*

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. Visit us at www.aestheticintegration.com

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

Copyright © 2015 Aesthetic Integration Limited. All rights reserved.

This document is written for information purposes only and serves as an overview of services and products offered by Aesthetic Integration Limited and/or its affiliate companies. References to companies and government agencies do not imply their endorsement, sponsorship or affiliation with services and products described herein. Aesthetic Integration, Imandra and 'The Logic of Financial Risk' are trademarks of Aesthetic Integration Limited. Imandra includes all or parts of the Caml system developed by INRIA and its contributors. FIX is a trademark of FIX Protocol Limited. SIX Swiss Exchange is a trademark of SIX Swiss Exchange Limited.

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.

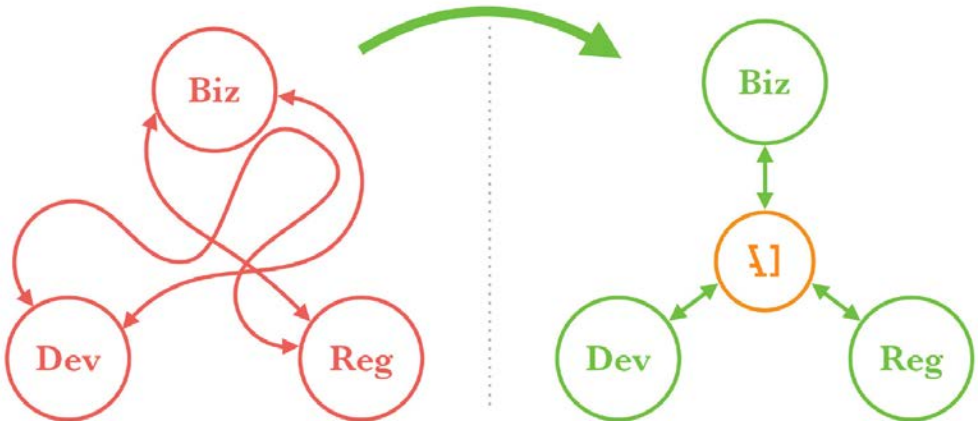

FIGURE 1: IMANDRA TRANSFORMING THE PROCESS OF CREATING TRADING SYSTEMS

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.

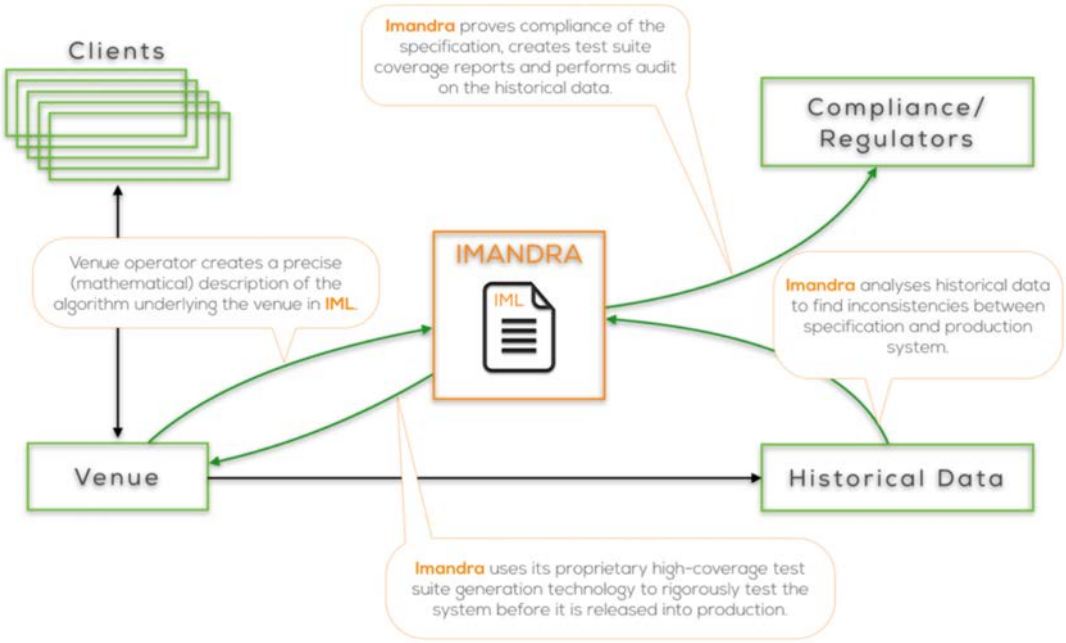

FIGURE 2: AUTOMATING COMPLIANCE WITH IMANDRA

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.

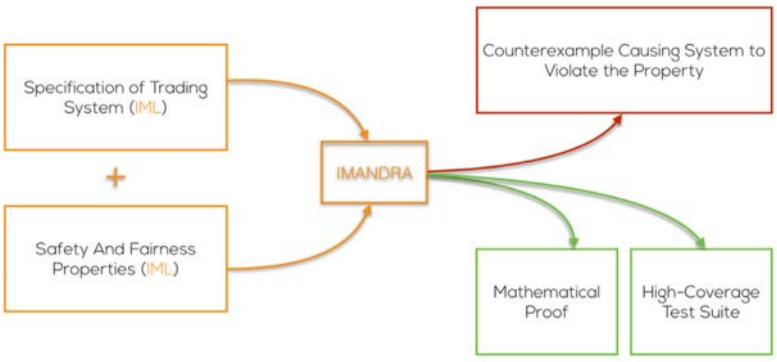

FIGURE 3: IMANDRA OVERVIEW

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 CONDITIONS

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: CONDITIONING TRADING ON CURRENT MARKET

Proving 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 8 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 TRANSITIVITY

```
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.qty > o2.qty then true
                   else if o1.qty < o2.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 = {
       peg = NEAR;
       order_type = PEGGED;
       qty = 1800;
       price = 10.5;
       time = 237;
       ...
       };
```

```
order2 = {
       peg = NEAR;
       order_type = PEGGED_CI;
       qty = 8500;
       price = 12.0;
       time = 237;
       ...
       };
```

```
order3 = {
       peg = NEAR;
       order_type = PEGGED_CI;
       qty = 8400;
       price = 10.0;
       time = 236;
       ...
       };
```

FIGURE 12: COUNTEREXAMPLE TO ORDER RANKING TRANSITIVITY

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 (in this case market bid is 10.0), 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, 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

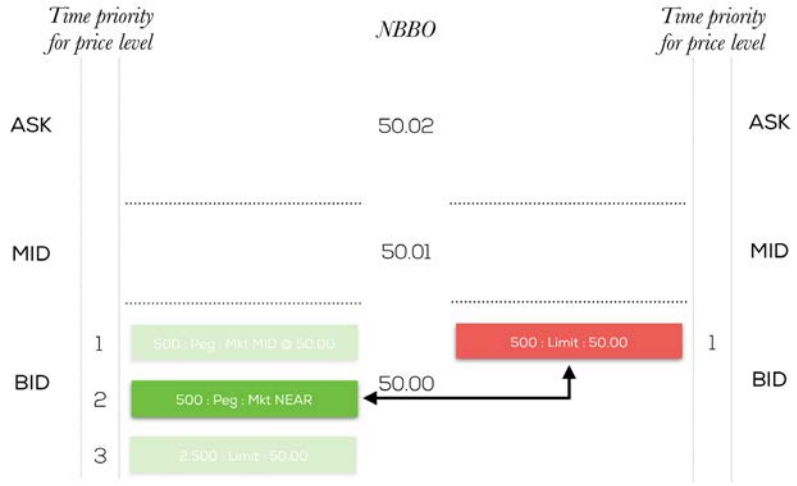

FIGURE 16: PegLimitConstraintMode = 2

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

Copyright © 2015 Aesthetic Integration Limited. All rights reserved.

This document is written for information purposes only and serves as an overview of services and products offered by Aesthetic Integration Limited and/or its affiliate companies. References to companies and government agencies do not imply their endorsement, sponsorship or affiliation with services and products described herein. Aesthetic Integration, Imandra and 'The Logic of Financial Risk' are trademarks of Aesthetic Integration Limited. Imandra includes all or parts of the Caml system developed by INRIA and its contributors. FIX is a trademark of FIX Protocol Limited. UBS is a trademark of UBS AG.