AESTHETIC AI INTEGRATION

TRANSPARENT ORDER PRIORITY AND PRICING

Transparent Order Priority and Pricing

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

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 very difficult to answer by looking at post-trade data alone. Fortunately, recent scientific breakthroughs allow us to systematically analyse algorithms for compliance and conformance with 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 those algorithms can and cannot do.

In this report, we illustrate Imandra's application to 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 to these issues lies in the application of modern scientific methods developed precisely for analysing behaviour of complex algorithms. Such methods have already become the backbone of engineering processes in other safety-critical industries such as avionics and hardware manufacturing. Most recently, they have proved themselves indispensable in the design of collision avoidance kernels within self-driving cars.

In the following report, we demonstrate how our product Imandra can be used to analyse behaviour 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 look at an order book on an exchange and you see limit orders - that snapshot of the data contained within the venue represents a state. If you send an order to the exchange and part of it crosses while residual rests on the order book, that's 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 there? 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 their 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 finally 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 obtuse mathematics involved.

Imandra

Imandra makes it possible to ask deep questions about an algorithm's possible behaviours, and to analyse implementations for conformance to their design. Such questions are encoded as *verification goals* (VG's). To reason about VG's, Imandra employs powerful patent-pending 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 configurations and inputs to the system. In the case 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. Those test suites can 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 can be automatically translated by Imandra into mathematics. Imandra's reasoning engine can 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 will contain details about 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 we would like to reason about. For example: *Does the venue accept 'sub-penny' orders*? In this report, we will demonstrate application of Imandra to two VG's 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.

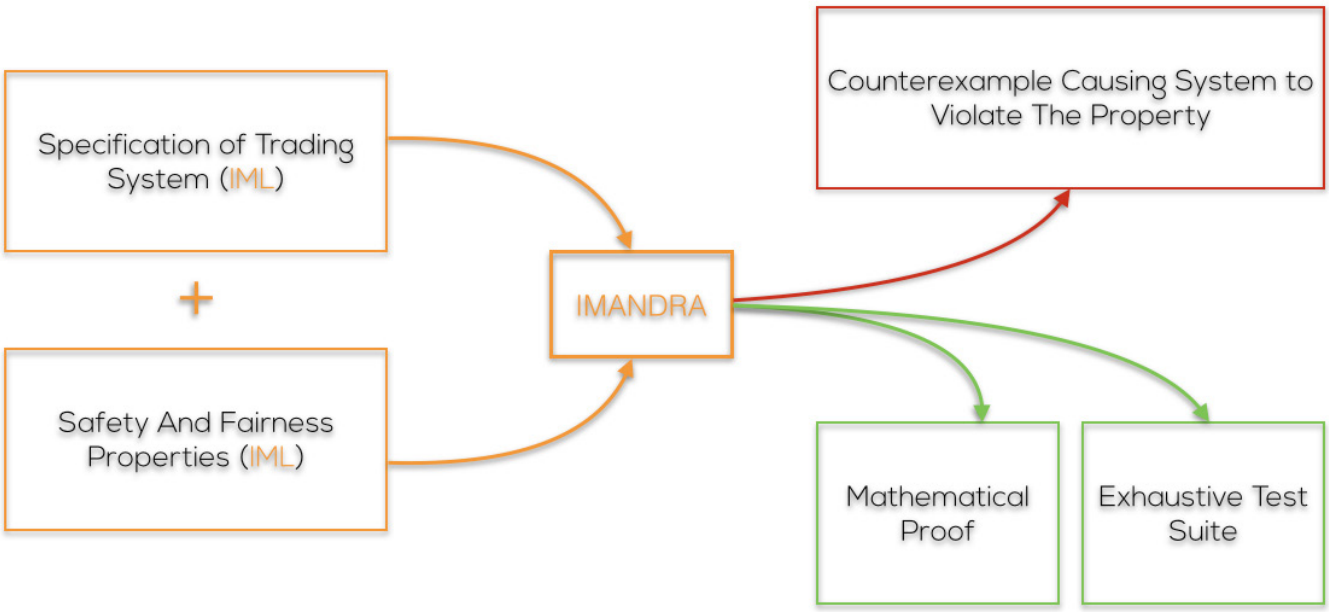

Figure 1: Overview of Imandra.

[1] IML's executable fragment is a subset of the programming language OCaml. 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 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)`'), and it will return `true`. This is nothing new; it's just programming. Now, what is special about Imandra is that not only can you write regular programs, but you can 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 throw in 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 Specification

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.

The two examples we describe 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 was 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 one of those order types.

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/or operator settings.

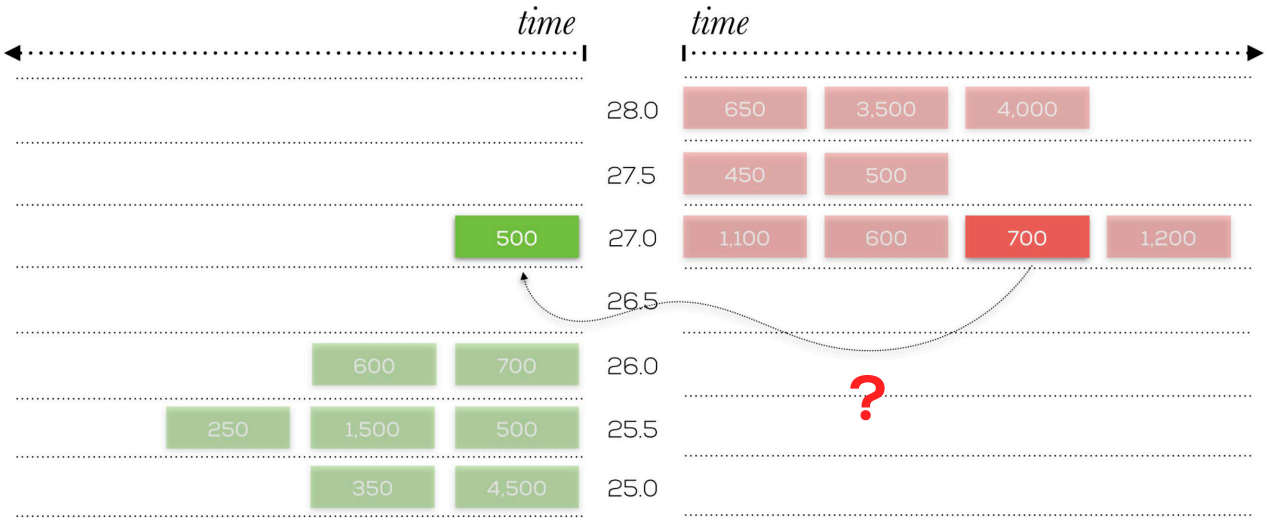

Figure 2: Out of priority, order to sell 700 matches against incoming market buy order for 500.

[3] You can compile any IML model into an 'executable simulator' and run it just like any other program.

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 can affect how an order is prioritised and traded. For example, consider logic within a market data handler. There are many moving parts. 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).*

Let's proceed to writing down actual IML code. We'll start by saying there are two orders (on the same side), but order 1 is more aggressive and has an older time stamp than order 2. 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 (this depends on order type and side). 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 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: *it allows you to encode high-level properties of the behaviour of an algorithm and automatically reason about it*. Such a verification goal is applied to the model of an entire venue, including the various risk gates, market data handlers, etc. It is a universal statement that can be applied to almost any venue, regardless of order types it supports or the number of client categories it has. Is it the only way to define such a property? Absolutely not - we leave exact formulations of the VG to the regulators and the industry to work on this together. Our purpose is to create a scientifically based and rigorous *medium* for expressing algorithm constraints and analysing algorithms with respect to them.

What can lead to a violation of this VG? The answer is: 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 will showcase the quite often surprising results of applying formal verification techniques to non-trivial systems.

Order Pricing

Imandra's Information Flow Analysis allows you to analyze and isolate 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 with 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.

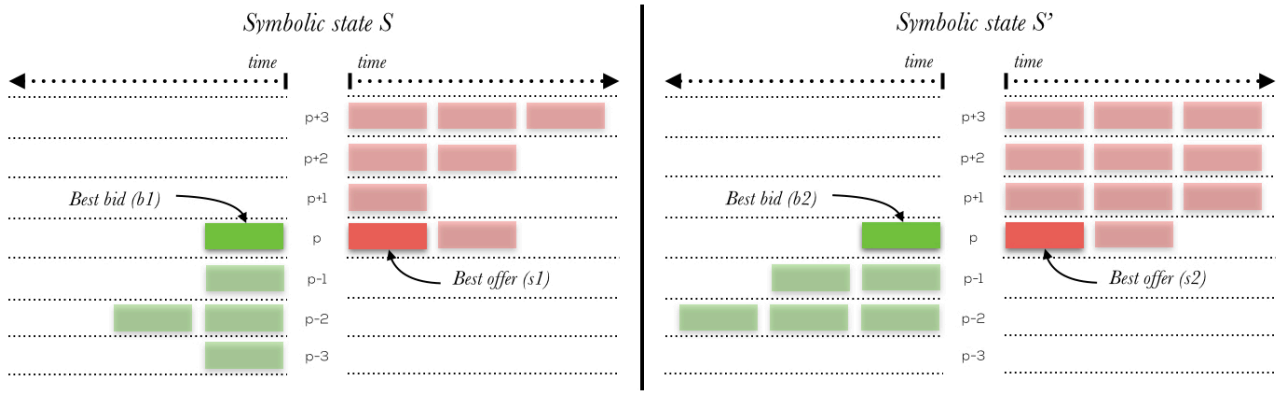

Figure 3: Our initial attempt to setup the verification goal.

Here's 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 scenarios are exactly the same (i.e. quantity, limit price, order type, etc.) **except** for their client IDs.
- Similarly with the best offers (they are equal in all attributes **except** for their client IDs).
- Furthermore, S and S' are equivalent in all state variables **except** for their order books.

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

[4] Recall that there is a virtually infinite number of possible states. By a *symbolic state* we mean that it may be any possible state, just as you might use the symbolic variable 'x' to mean 'any real number' when doing an algebraic calculation.

Here's 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 will result from 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
```

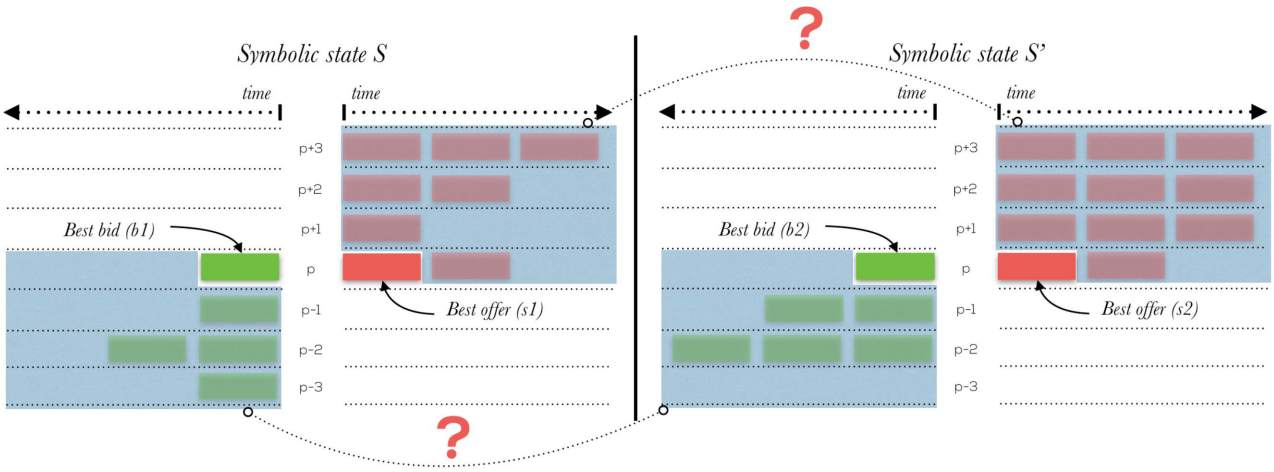

Figure 4: The counterexample pointed out a flaw in how we formulated our verification goal.

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's 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')
```

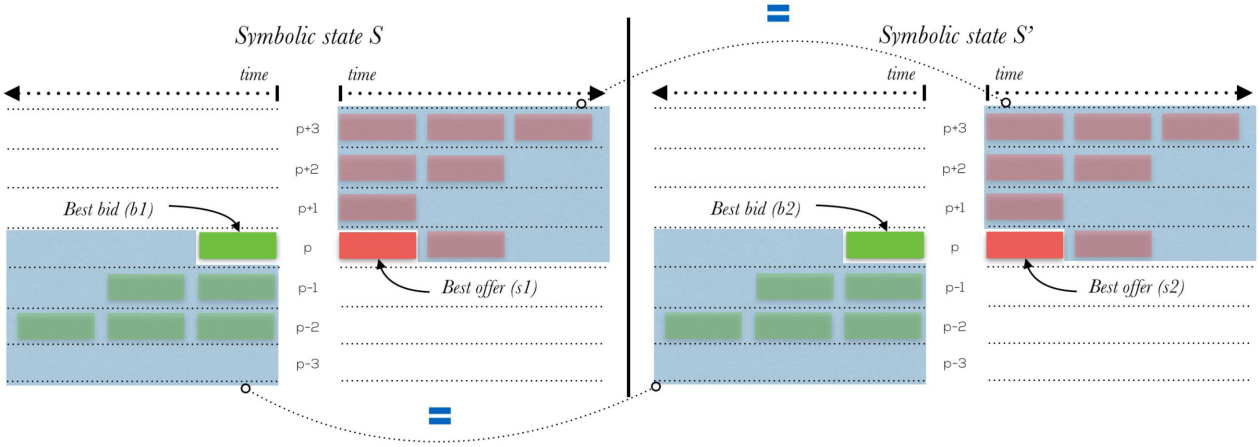

Figure 5: We strengthen our verification goal by placing additional constraint on 'tails' of the order books.

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 the verification goal (i.e., Imandra proves the VG), you can start to reason about whether the production system is faithful to the specification.

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.

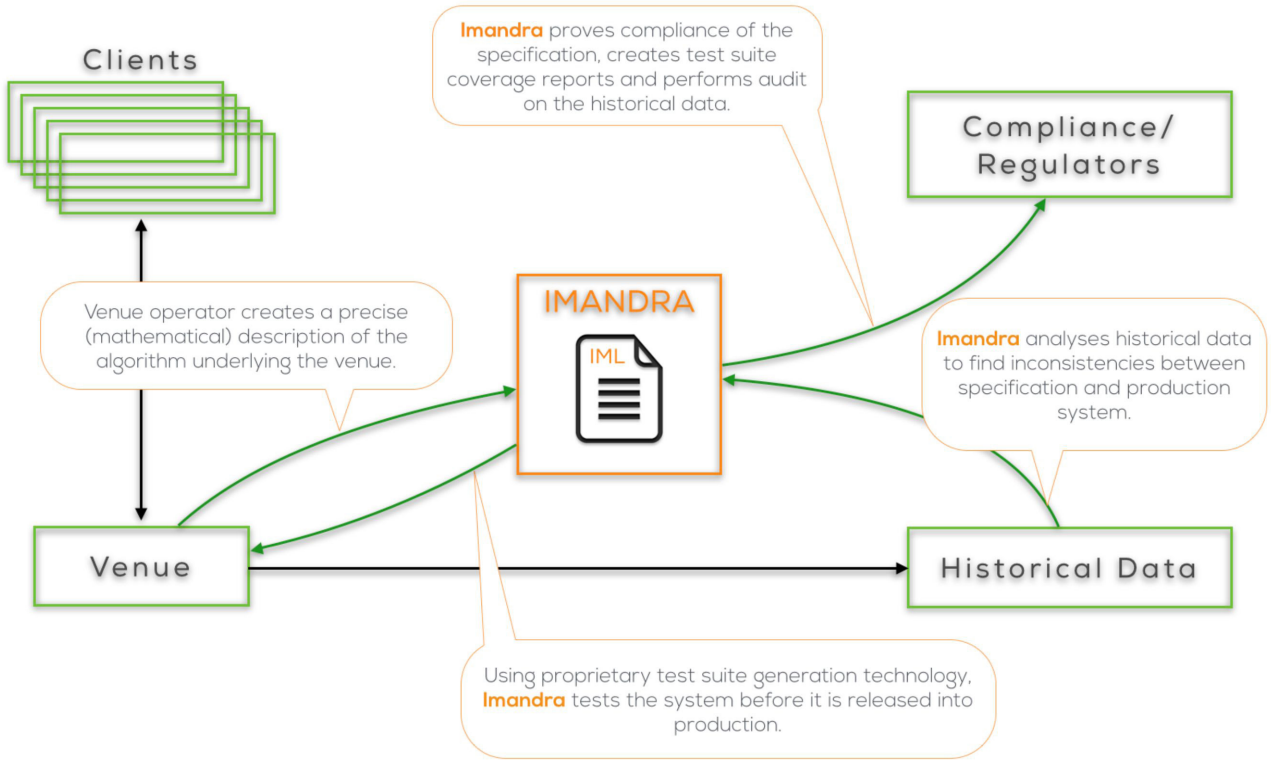

Figure 6: Integrating Imandra into design/development process.

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 other 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