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 the process of 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.

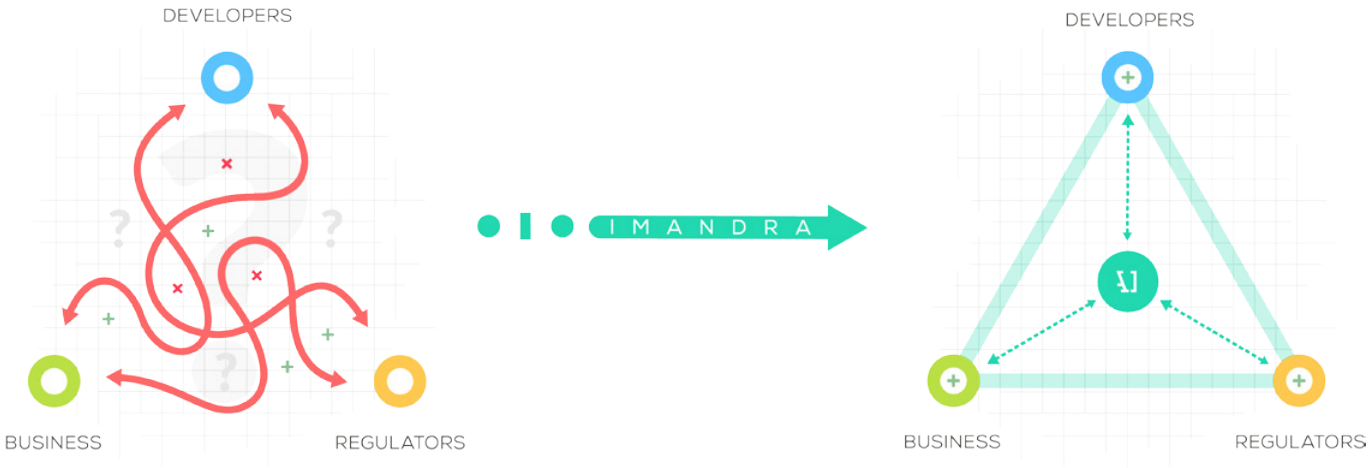

FIGURE 1: IMANDRA TRANSFORMING THE PROCESS OF CREATING TRADING SYSTEMS

In a typical investment bank, the process of designing, implementing and regulating trading systems and venues 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.

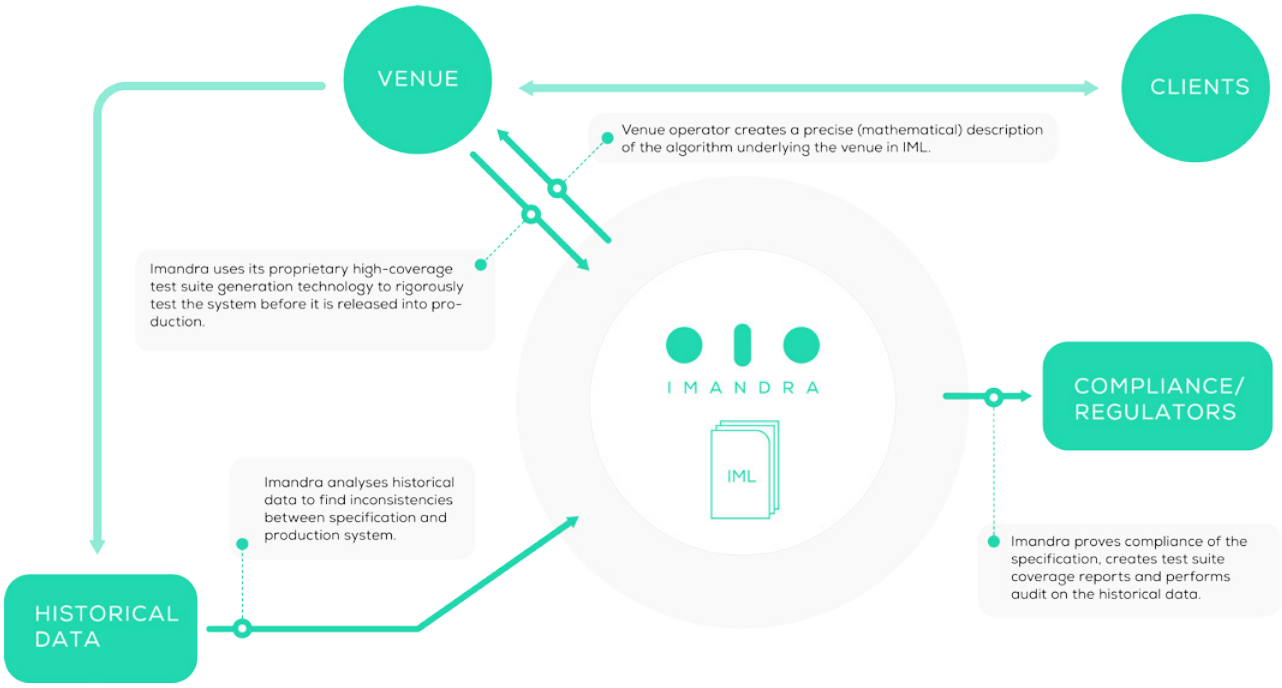

FIGURE 2: AUTOMATING COMPLIANCE WITH IMANDRA

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 the building he/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: Venue developers will have a precise specification of the functionality - this will cut down time needed to understand business requirements. Quality Assurance will have tremendous power with Imandra's automated test suite generation enumerating logical corner cases. Those responsible for systems that send orders to venues can query the Imandra specification to answer any questions they might have about how the venue will communicate with their system. Contrast such approach with having to decipher ambiguous PDF documents and marketing materials.

Regulatory functions (compliance officers): With Imandra, compliance officers can encode 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 our product, Imandra, can significantly assist in designing, implementing and regulating modern trading systems and venues. With Imandra, UBS can leverage major breakthroughs in algorithm analysis ("formal verification") to help ensure its 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 radically 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 Imandra Model of UBS ATS

Our complete Imandra model of UBS ATS (as described in the referenced Form ATS) is about 800 lines of IML code. In terms of the workload involved in creating it, we expect it should not take more than two weeks to encode 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 ordered 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 meaning to these order types, but 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 - Imandra will not accept the model as complete unless the user described how orders are priced for each of those declared order types.

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
      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
  (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, etc.

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

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

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

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

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

Order Priority Rules

Our last example demonstrates application of Imandra to reasoning about order prioritisation rules. This example is motivated by numerous debates about 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 can ensure 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 should make 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.

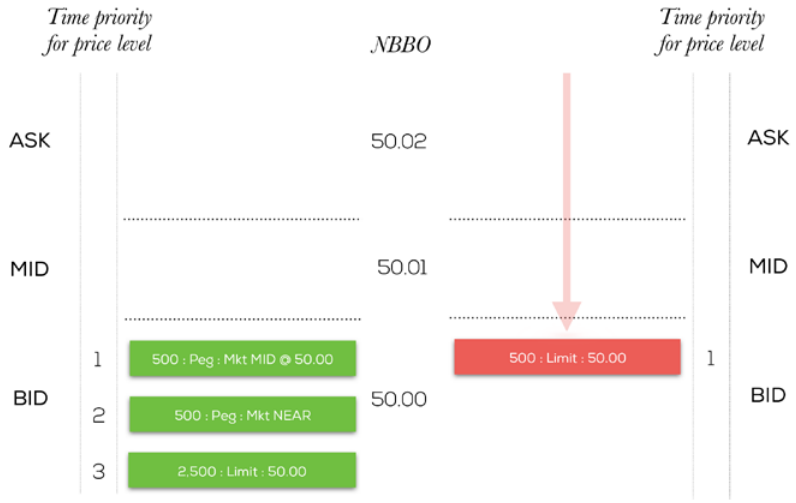

FIGURE 14: ORDER BOOK HAS AN INCOMING SELL ORDER

FIGURE 15: PegLimitConstraintMode = 1

FIGURE 16: PegLimitConstraintMode = 2

It 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