AESTHETIC AI INTEGRATION

TRANSPARENT ORDER PRIORITY AND PRICING

AUGUST 2015

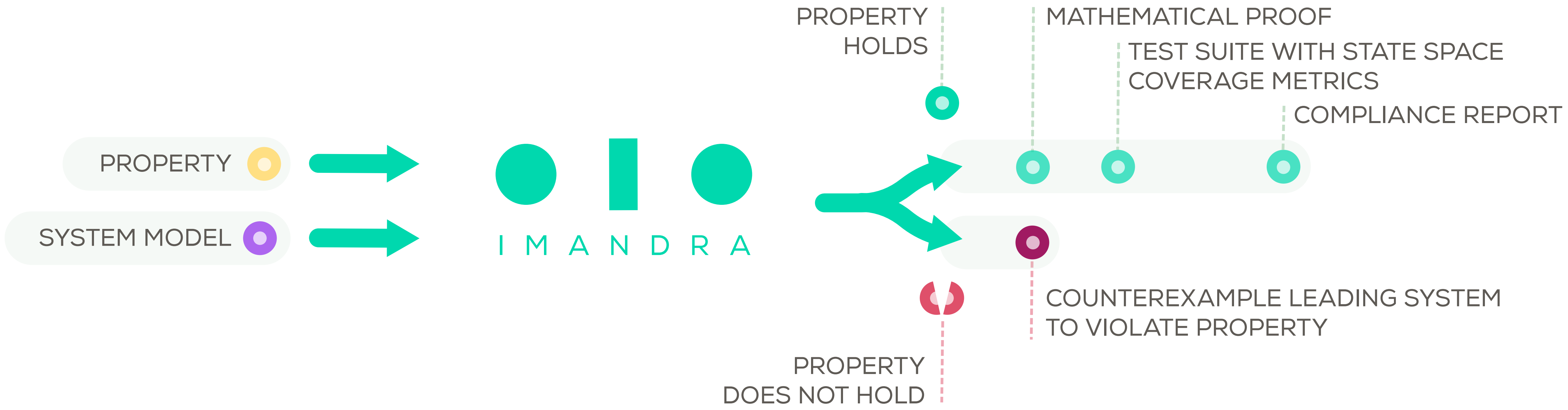
Outline

The Problem Statement

- ❖ Financial markets run on complex trading systems that process tremendous volumes of data at lightning speeds.
- ❖ Venues (e.g., dark pools and exchanges) operate complicated algorithms with numerous order types, circuit breakers, trading regimes, etc.
- ❖ Such complexity leads to virtually infinite system state-spaces (collections of all possible configurations and behaviours of the system).
- ❖ To make matters worse, the communication format for disclosing venue matching logic is typically ambiguous English prose.
- ❖ *As a result, flaws (intended and unintended) in design and implementation of venue matching engines are notoriously difficult to detect.*

Solution: Imandra by AI

- *Imandra is an automated solution for reasoning about the design and implementation of complex trading algorithms and systems.*
- It allows you to *mathematically prove* compliance of your system with regulatory directives and consistency with marketing materials.
- **Imandra sets the new standard of transparency for financial algorithms.**



Where Complexity Comes From

- *Q: What is a 'state-space' and why is it so big?*
- A: A *state* of a system is one of its possible configurations. A state-space is a mathematical description of all possible configurations of a system, and how they relate to each other. Modern venues have virtually infinite state-spaces. The support for many order types (with various attributes), transition between different trading periods and numerous regulatory (e.g. Reg NMS) directives makes venue state-spaces extremely complex.
- *Q: How does Imandra manage this complexity?*
- A: Imandra leverages recent advances in formal verification to automatically convert venue specifications into mathematical logic, and applies powerful automated theorem proving techniques to analyse venue behaviour. It further leverages this analysis to derive high-coverage test suites to test high-performance venue implementations for consistency with their design.

Creating Precise Specification in IML

- In order to understand whether a venue does what it is supposed to do, we must first precisely describe its design.
- With the Imandra Modeling Language (IML), declarative statements about order types are precise:

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

- And so is the meaning assigned to them. Here's a fragment of an IML venue model used in calculating the price at which an order will 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
```

- Information disclosed within Form ATS can be easily encoded in IML.
- Imandra comes with many libraries of generic types of trading systems - you only have to specify features that are specific to your venue!

What is a Verification Goal?

- A *verification goal* (VG) is a *statement about a program* that is either true or false. Formally, a VG is a function whose output is a boolean value. When Imandra *proves* a VG, it actually proves it will evaluate to true for *all possible inputs*.

- As an example, let's consider a trivial function:

```
let simple (x) = x > 5
```

- `simple` returns `true` if its input variable x is strictly greater than 5 and `false` otherwise. So, `simple(6) = true`.

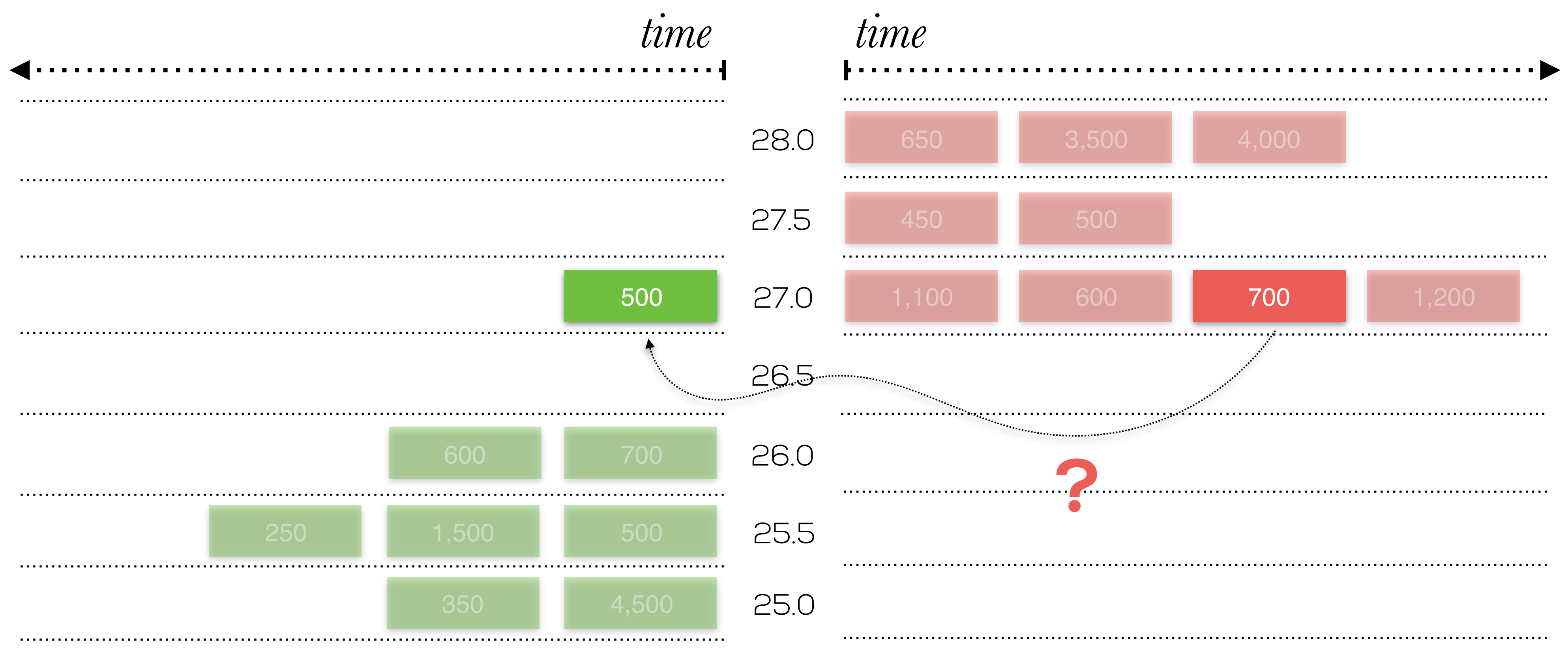
- This is nothing new - it's just programming. However, with Imandra, you can automatically reason about `simple`'s behaviour:

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

- `simple_VG` is a verification goal that says that if `x` (the input into `simple`) is greater than `10`, `simple(x)` will **always** return `true`.

- Our next two examples will use IML specifications of venues and describe VGs dealing with order priority and pricing.

Order Priority Rules



What are the typical factors influencing an order's priority in the queue?

- *Price* - price at which an order is willing to trade. May depend on the order type (i.e. Market or Limit), limit price (if applicable), peg and NBBO.
- *Time* - order arrival timestamp
- *Category* - client category
- *Trading Constraints:* there are numerous other constraints (e.g., minimum quantity) that can potentially prohibit two orders from trading with each other

Order Priority Rules

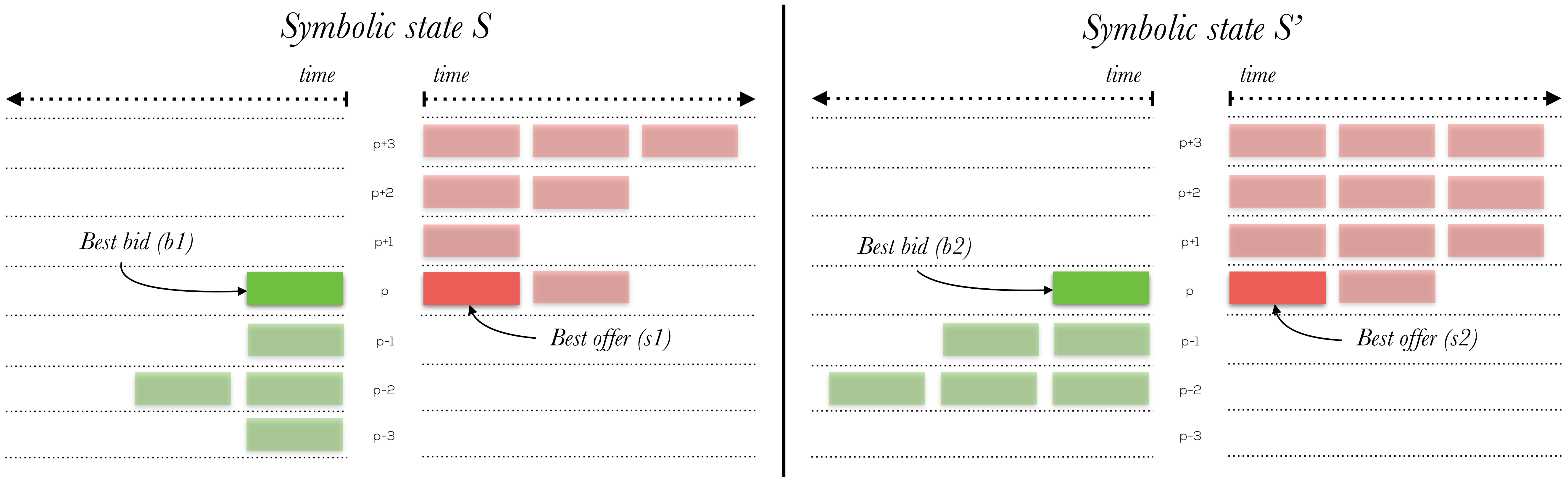
- Here's the motivation in "plain English": *If I send an order to the venue and my order is the more aggressive and it's older than another order there, then I should get filled first (unless there are restrictions that prevent a trade that are disclosed to me in the marketing materials).*
- Here's 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'));;
```

- Some comments on the functions that get called:
  - `s' = next_state(s)` means that after matching all possible orders, processing messages, etc., the current state `s` will transition to `s'`. The new state variable will contain all executed fills.
  - `order_at_least_as_aggressive` calculates the most aggressive prices at which orders 1 and 2 are willing to trade, and returns `true` if order 1 is at least as aggressive (side-dependent) as order 2.
  - `first_to_trade` - if there's a fill for order 2, then there must be a fill for order 1 (our order).

Order Pricing: Initial Approach

- Our next example is an application of Imandra's Information Flow Analysis (IFA) to fill pricing
- The "plain English" motivation for the VG: *Client ID should not play a role in the calculating price of a fill.* There are many ways to encode this in IML.
- The setup: Let's take two symbolic states *S* and *S'* of the exchange where tops of the books (i.e., best bids and offers) are exactly the same, except for client IDs. The two states are exactly the same except for their order books.
- We would then expect two the best bid and offer to trade at exactly the same price for both of these scenarios.



Order Pricing: Initial Approach

- The resulting 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');;
```

- As in the previous example, we provide details on functions called within the verification goal:
  - `orders_same_except_source` compares two orders and returns `true` if and only if they are exactly equal except for their source (and `false` otherwise).
  - `states_same_except_order_book` compares two venue states and returns true if and only if all of its members except for the order books are equal. This includes the time of day, current period, unprocessed/processed messages, fills, etc.
  - `best_buy` and `best_sell` both take the venue state and return the best bid and offer respectively.
  - `match_price` takes the venue state and returns the fill price of the next trade (if any two orders can trade).
  - `==>` is an operator for *implies*. Definition: `a ==> b` is `false` when `a` is `true` and `b` is `false`, and `true` otherwise.

Order Pricing: Counterexample

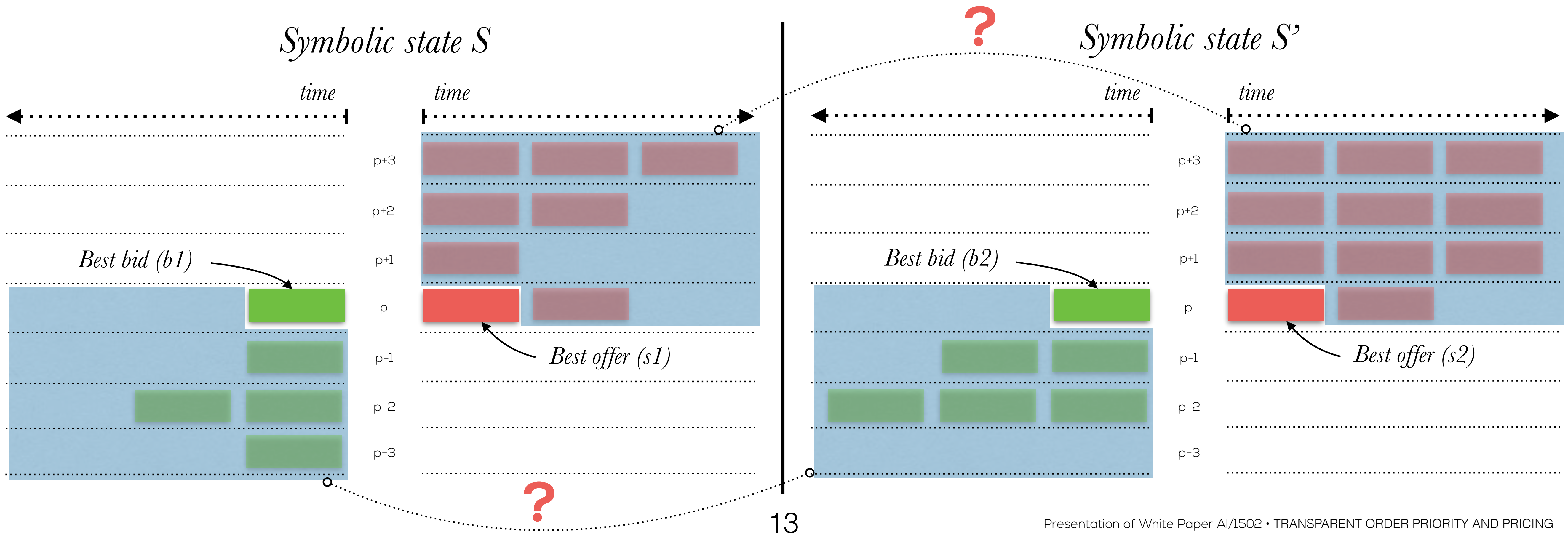
- When we ask Imandra to reason about the verification goal, it quickly returns a counterexample - inputs into the system leading it to violate the VG!
- This is quite normal for formal verification - often things we *assume* to be true exhibit surprising behaviour around the 'edges'.
- *Please note that we are not printing out the entire state value here, only the relevant portion*

```
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
```

Order Pricing: Counterexample

- The counterexample highlighted a flaw in the original verification goal!
- Our initial and naive formulation of the VG did not account for orders that followed the best bid and offer. The counterexample highlighted that when both best bid and offer are `MARKET`, then the venue must transition into the `Auction` phase.
- According to the rules of the exchange, the auction uncrossing price will be based on the maximum volume matched. Hence, other orders (besides the best bid and offer) may influence the price.



Order Pricing: Adjustment

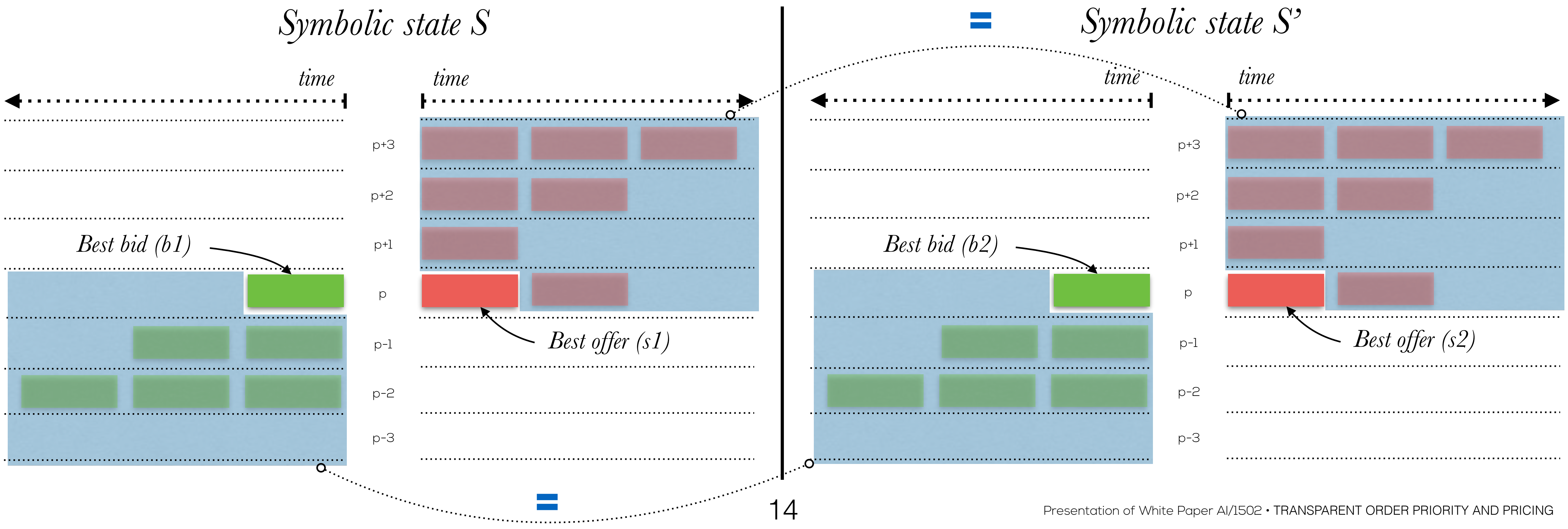
- Let us now update the VG to take into account orders after the best bid/offer.
- Here are the additional constraints we will add:

```
&& List.tl s.order_book.buys = List.tl s'.order_book.buys
&& List.tl s.order_book.sells = List.tl s'.order_book.sells
```

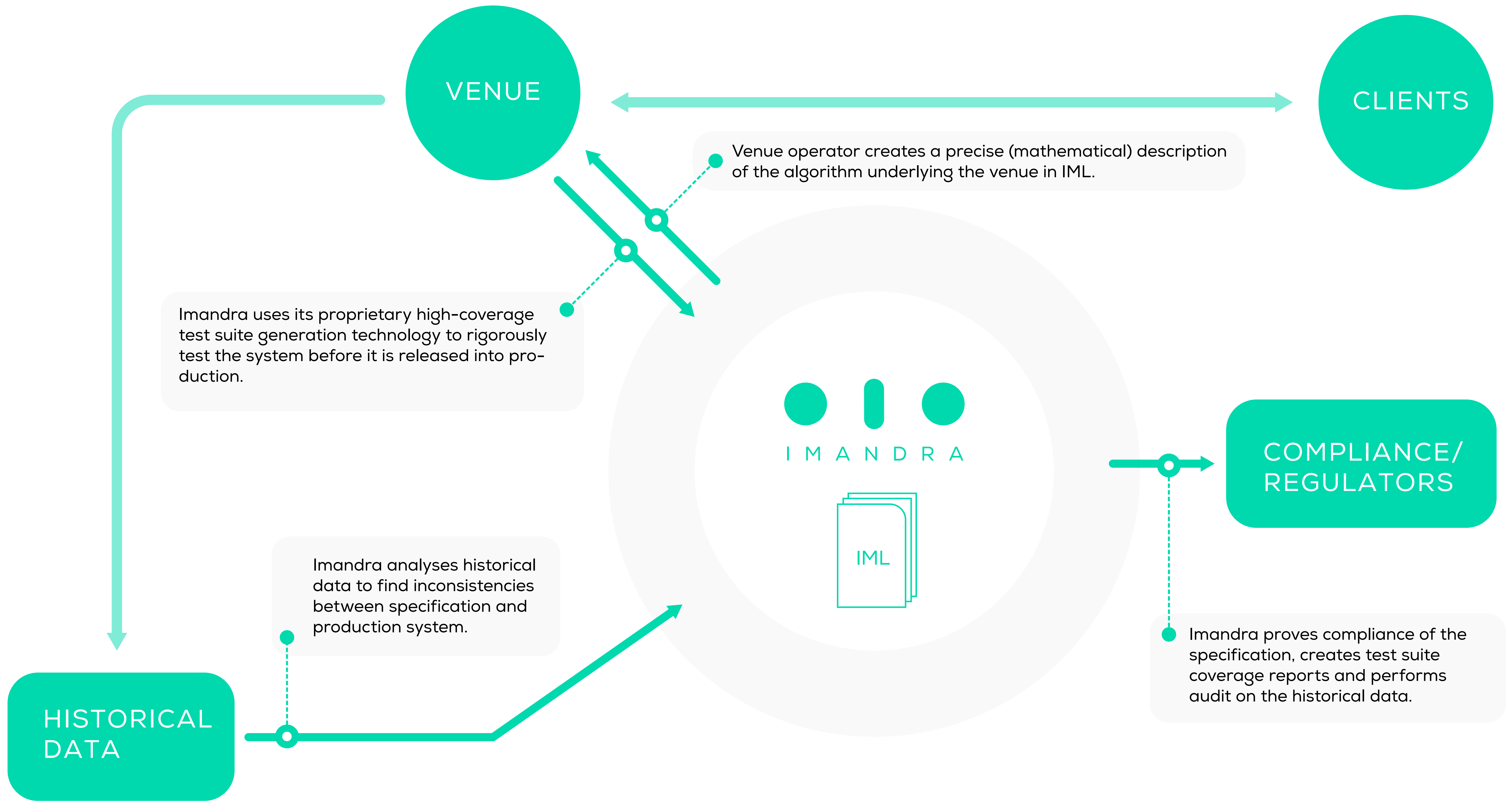
- When we ask Imandra to reason about the updated VG, it comes back with:

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

- We can further ask it to provide the actual proof object and submit it to third parties.



Automating Compliance With Imandra



Venue operator creates a precise (mathematical) description of the algorithm underlying the venue in IML.

Imandra uses its proprietary high-coverage test suite generation technology to rigorously test the system before it is released into production.

Imandra analyses historical data to find inconsistencies between specification and production system.

Imandra proves compliance of the specification, creates test suite coverage reports and performs audit on the historical data.

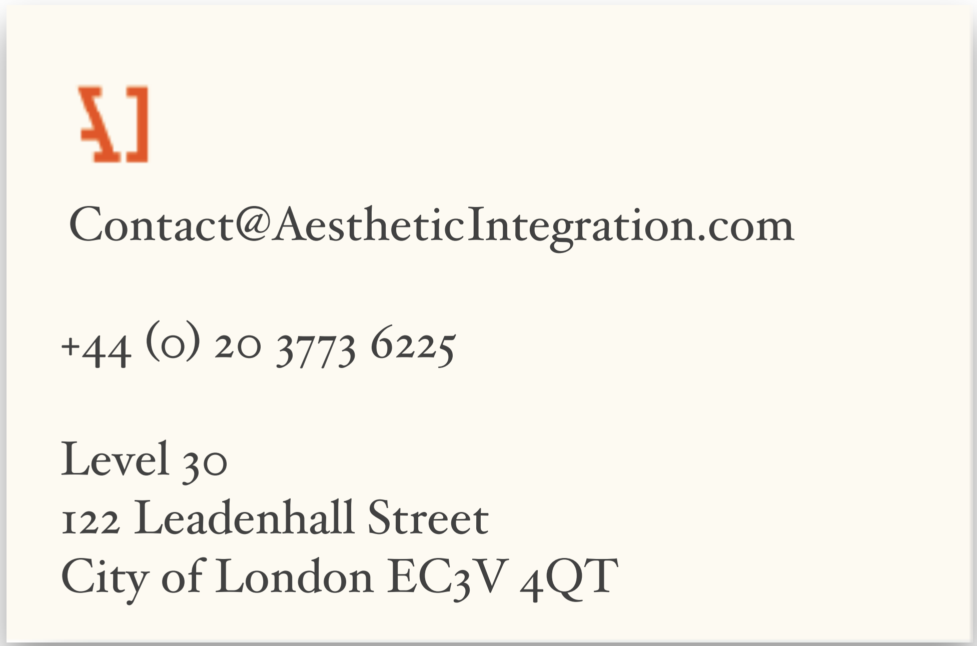
Beyond Order Priority And Pricing

Several other examples of VGs that you can reason 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?

Contact

AESTHETIC AI INTEGRATION


Contact@AestheticIntegration.com

+44 (0) 20 3773 6225

Level 30
122 Leadenhall Street
City of London EC3V 4QT

Legal Notice