AESTHETIC INTEGRATION

# CREATING SAFE AND FAIR MARKETS

# Creating Safe and Fair Markets

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

## Abstract

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

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

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

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

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

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

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

## I. MANAGING THE INFINITE

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

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

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

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

To gain some intuition, consider a sorting algorithm

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

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

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

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

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

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

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

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

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

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

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

## II. HOW OTHER INDUSTRIES DEAL WITH COMPLEX ALGORITHMS

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

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

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

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

## III. INTRODUCING IMANDRA

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

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

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

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

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

Imandra has the following key properties:

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

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

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

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

## IV. REGULATORY OVERSIGHT

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

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

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

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

### IV.1 Designing Directives

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

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

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

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

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

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

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

## IV.2 Quantifiable Testing Standards

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

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

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

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

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

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

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

## IV.3 Linking Regulation With Financial Economics

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## CLOSING REMARKS

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

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

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

## REFERENCES

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## APPENDIX: INDUCTIVE PROOFS OVER DATATYPES AND ALGORITHMS

### I. Mathematical Induction

Consider a computable monadic predicate

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

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

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

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

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

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

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

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

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

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

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

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

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

**Theorem 1** (Gauss).

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

*Proof.*

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

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

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

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

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

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

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

## II. List and Tree Induction

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

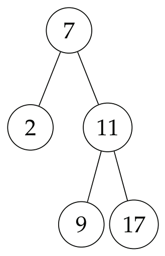
As a final example of structural induction, let us consider a datatype of binary trees defined as follows:

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

For example, the following is such a tree:

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

which might be visualised as:



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

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

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

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

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

## III. More Powerful Forms of Induction

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

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

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

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

$$\omega^{\beta_1} c_1 + \omega^{\beta_2} c_2 + \ldots + \omega^{\beta_k} c_k \text{ s.t. } k \in \mathbb{N},\ c_i \in \mathbb{N}^+,\ \beta_i \in On \text{ and } \beta_1 > \beta_2 > \ldots > \beta_k \geq 0,$$

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

**Ordinal Arithmetic:**

$$\alpha + 0 = \alpha$$

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

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

$$\alpha \cdot 0 = 0$$

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

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

$$\alpha^0 = 1$$

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

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

**Induction principle:**

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

Many powerful automated reasoning techniques exist for well-founded induction, especially those due to Boyer-Moore [3] and found within the ACL2 theorem prover (for quantifier-free induction up to the ordinal $\epsilon_0$) [11]. With recent advances in automated model construction, these techniques can be significantly strengthened, e.g., by using (non-standard) counterexamples to guide nuanced forms of inductive generalisation. Imandra's automated induction builds upon these many advances.

## ABOUT AESTHETIC INTEGRATION

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

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

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

### Founders

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

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

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

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

*See our website for details: www.aestheticintegration.com*

### Contact

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

### Legal Notice