(ρro)Metheus: Provenance Resolution Oracle for Covenant-Bound Artificial Intelligence

Ling Xiao. [1]
Metheus

Abstract

This technical document introduces the (ρro)Metheus Oracle, a protected computational environment whereby analysts mobilize:

1. audited artificial intelligence (AI) models,
2. validated information pipelines,
3. to compute *causal relations* from data.

Although (ρro)Metheus is a general purpose computational oracle, its initial deployment targets are the regulators, central/commercial banks, insurers, and other entities serving transnational trade in the Global South. Specifically, outputs of (ρro)Metheus will inform high-valued decisions pertaining to the disbursement of funds and claim resolutions. These are business-critical decisions that require both:

- regulator oversight *a priori* oracle deployment;
- and proper assignment of liabilities *a posteriori* oracle judgment.

This implies every step of the computation inside the (ρro)Metheus Oracle must be properly governed by state-level entities, so that the oracle satisfies the strong guarantee of:

- truthful or compliant computation runs to completion;
- while illegal or false computations fail outright.

This requirement is addressed by statistically-typed programming languages, where programs that do not terminate fail at compile time. We enforce this design pattern in (ρro)Metheus: at the heart of the oracle is a *domain-specific programming language* called: Typed ρ-Calculus for covenant-bound probabilistic causal reasoning. It is a two-tiered computational model whereby:

- *in the term universe* or the "analytic" level, (ρro)Metheus uses Pearl's Go-Calculus to evaluate causal relations. In particular, the oracle uses information from validated data sources and summaries generated by audited large language models (LLMs) to compute causal proofs.
- *at the type universe* or the compliance level rests a meta-language extended by state regulators. The type-system governs the quality or legality of computation, so that illegal and false computations terminate immediately. Thereby providing a static guarantee that (ρro)Metheus-bound AI agents are well-behaved in accordance to legal constraints.

When verifying programs for contractual compliance, the (ρro)Metheus compiler uses a legalistic analogue of Floyd–Hoare logic, or Loare-Logic to generate proofs witnessing the Covenant-Compliance of Typed ρ-Calculus code written by analysts. This automated proof system shifts the burden of writing legally-compliant programs from analysts to the (ρro)Metheus compiler, thereby extending the operation domain of AI agents into more sensitive and regulated settings.

In other words, if *code is law*, then one may conceptualize AI agents as entry-level employees within enterprises, so that the (ρro)Metheus sandbox is populated by a set of *law-abiding* junior employees. Meanwhile the (ρro)Metheus type-universe is the *constitution*, or the "system Loare." This Loare is the *lingua-franca* of data and AI validity in the domain of international trade, and shall sit in the public domain as a global public good. We brand this computational model distinguished by 1) validated data pipelines, 2) audited AI models, and 3) deep regulator oversight governing the computation graph of the (ρro)Metheus oracle: Covenant-Bound Artificial Intelligence.

[1] lingxiao@metheusai.xyz

Keywords and phrases Machine learning, statistics, artificial intelligence, formal verifications, deep AI governance, AI safety, programming language design, domain specific programming language, legalistic technology.

Contents

1 Introduction

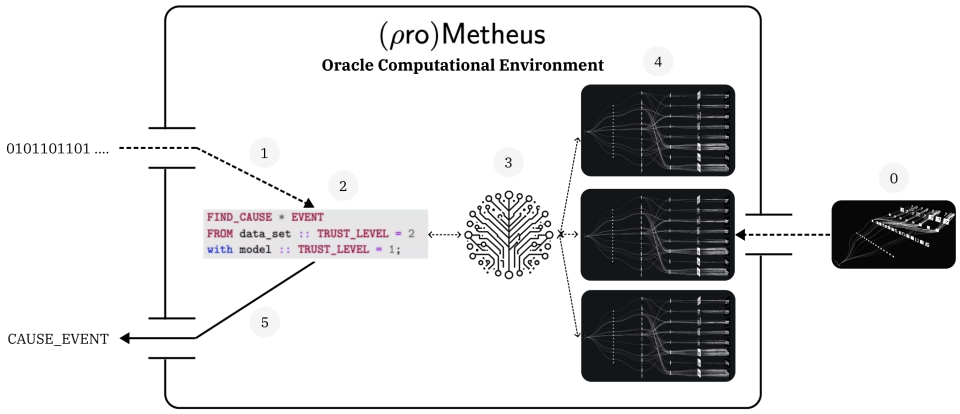
1.1 Technical Stakeholder Perspective

This is a system specification document for the (ρro)Metheus **Provenance Resolution Oracle** for Covenant-Bound Artificial Intelligence. It features:

- A *regulator-compliant* computational environment that enables analysts to run *audited* machine learning (ML) models on *validated* datasets. [2] We refer this set of ML models and datasets as *computational assets.*
- The (ρro)Metheus oracle is especially suitable for recovering casual relationships in datasets in a statistically rigorous manner.
- The theoretical underpinnings of the (ρro)Metheus Oracle is a domain specific programming language named: Typed ρ-Calculus for covenant-bound probabilistic causal reasoning. It combines decades of research and design in programming language theory, classical statistics, and recent advances in machine learning to produce a safer way of interacting with existing AI agents.

1.1.1 Computational Asset Lifecycle in (ρro)Metheus

In concrete terms, (ρro)Metheus is a user interface targeting industry professionals at banks, insurance companies, and regulators.



This user interface is the visible face of the (ρro)Metheus protected computational environment, whereby analysts interact with audited computational assets. The asset lifecycle proceeds as follows:

- (0,1): model and data are audited using stakeholder-defined rules of validation, and subsequently assigned a type, or more precisely a Covenant-Judgement. These legally-tagged computational assets then enter the (ρro)Metheus environment.
- (2) analysts query the data lake with Typed ρ-Calculus in lieu of an LLM-based chat assistant. In particular, they query data with the intention of computing the cause of some real-life **EVENT**.
- (3) (ρro)Metheus compiles the Typed ρ-Calculus language and mobilizes legally-compliant class of data and machine learning models to complete the task.
- (4) (ρro)Metheus draws on open-source and bespoke ML models to compute the likelihood of different causes of the **EVENT**.
- (5) (ρro)Metheus returns probable causes of this **EVENT** to the analyst.

[2] See definition 1 for more on computational environment.

In the figure above, the programming language Typed ρ-Calculus is the top-level user interface. Alternatively we may explore other user interfaces such as:

- A graphic user interface, which may be more appropriate in a consumer setting.
- A natural language interface using LLM's code-generation feature. In this case, we train an LLM to learn a mapping from natural language onto Typed ρ-Calculus.

The advantage of (ρro)Metheus computational environment is three-fold:

- All data and models are audited in accordance to stakeholder-defined laws of validity and security.
- These rule governing validity of information are then propagated along Typed ρ-Calculus's type system from data collection, to model training, fine tuning, and finally inference.
- Additionally, Typed ρ-Calculus has rigorous notions of causality that cannot be expressed in the "classical statistics" underlying machine learning models. Classically trained agents such as auto-regressive models (read: LLMs) can only express correlation, not causation.

The heart of (ρro)Metheus Oracle is the domain specific programming language: Typed ρ-Calculus for covenant-bound probabilistic causal reasoning. Unlike classical statistic language, Typed ρ-Calculus can express causation in addition to correlation.

▶ **Definition 1.** *[7] A **Computational Environment** involves the collection of computer machinery, data storage devices, work stations, software applications, and networks that support the processing and exchange of electronic information demanded by the software solution.*

Next we expand on two use cases of (ρro)Metheus in particular:

- one that targets who query audited data using audited ML models and validated datasets;
- one that targets regulators who define the criteria of valid data and models.

Although they appear to be two orthogonal deployment scenarios, they are in fact part of the same pipeline that takes regulator's legal requirements, and translate them into a compliant artificial intelligence computational environment.

1.1.2 An Example for Analysts

The analysts at banks and insurance companies interact with (ρro)Metheus as a simplified programming language. The language of interface is Typed ρ-Calculus. It is akin to `MySQL`, however instead of rudimentary operations that simply query the database with:

```
SELECT *
FROM employees;
```

The Typed ρ-Calculus under proposal enables analysts to interact with their database in a categorically more sophisticated manner, by seeking the cause of events with:

```
FIND_CAUSE * EVENT
FROM data_set :: TRUST_LEVEL = 2
with model :: TRUST_LEVEL = 1;
```

In this code block, we assume `data_set` and `model` are assets that are pre-loaded in the (ρro)Metheus computational environment. Some additional commentary follows:

1. (line 1): note instead of merely selecting data as is the case with `MySQL`, the analyst uses Typed ρ-Calculus to query the database for the probable cause of `EVENT`.
2. (line 2): the setting `TRUST_LEVEL=2` in line 2 expresses the trustworthiness of the data used to determine the cost of `EVENT` in line 1. In this case `TRUST_LEVEL=2` is a *regulator*-defined trust setting on the validity of the data under consideration. It is effectively a filter on the quality of data used to determine the final cause of `EVENT`.
3. (line 3): the statistical models are also typed with `TRUST_LEVEL=1`, which determines the quality of statistical model used to determine the final cause of `EVENT`. This is critical as the quality of statistical models is a function of:
   - the quality of statisticians training them;
   - the rigor of the assumptions underlying the models;
   - and most importantly, the quality of data the model is trained on.

Naturally models trained on `TRUST_LEVEL=2` data can do no better than attaining a `TRUST_LEVEL=2` itself, as the adage: "garbage in garbage out" attests. [3] The proper propagation of `type` assignment from datasets to models trained on said datasets is a core feature of Typed ρ-Calculus. Now observe that in the Typed ρ-Calculus query above, the quality of dataset is lower than the quality of the model. Then intuitively the final `CAUSE` of `EVENT` determined by this short program should carry `TRUST_LEVEL=2`.

The interaction of datasets and models at inference time is an example of *program composition*. In programming language (PL) parlance, the *inferred type* of the final recommendation correspond to the confidence of the final recommendation in statistical terms. The intellectual core of Typed ρ-Calculus pivots upon this correspondence between type in the PL sense, and a promise or *covenant* of compliance in the legal sense. We term this category of legally-compliant and statically guaranteed artificial intelligence: Covenant-Bound Artificial Intelligence. This correspondence is explored in table (1).

> The intellectual core of Typed ρ-Calculus pivots upon the one-to-one correspondence of *covenant* in the legal sense, with inferred the *type* of data in the programming language sense.

1.1.3 An Example for Regulators

In the previous example, we assumed the criteria that determined which computational asset satisfies any particular `TRUST_LEVEL` was a given. In reality, this criteria is determined by regulators. This

[3] For the sake of simplicity we assume `TRUST_LEVEL=2` is less trustworthy than `TRUST_LEVEL=1`.

section provides an example of how regulators would interact with (ρro)Metheus to define the vectors of quality along which computational assets are audited against. Since a regulatory environment with broad stakeholder buy-in is seldomly unilaterally defined, finding consensus on the *criteria to audit* must be an interactive process. Now referencing figure (1), (ρro)Metheus presents a user interface whereby:

1. Multiple parties enter a common covenant proposal environment, so as to submit candidate criteria for community judgement.
2. What follows is an iterative process, whereby the stakeholders debate and elect the most sensible set of criteria to judge data validity and AI model quality.
3. Finally this criteria is reified in the (ρro)Metheus computational environment as a type or Covenant-Judgement.

One example of such a Covenant-Judgement is `TRUST_LEVEL` that will be used by the analysts in the example above. Finally observe that depending on the quality of the dataset or AI model, it may only witness some of the criteria defined in Covenant-Judgement. This fine-grained satisfaction is what give rise to different levels of `TRUST_LEVEL`, i.e. `TRUST_LEVEL` = 1 etc.

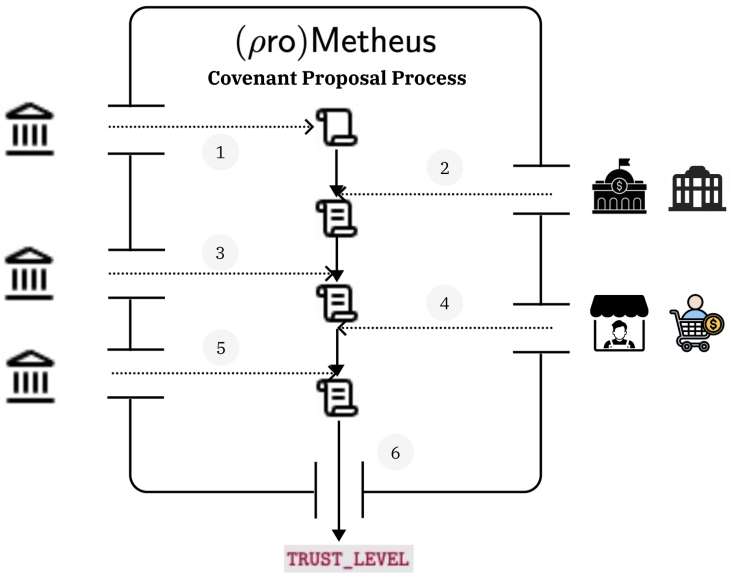


Figure 1 The (ρro)Metheus covenant proposal process is an iterative procedure where the regulator proposes criteria of audit in step (1). In step (2), other stakeholders such as financial institutions amend the proposal. This is sent to the regulator for review in step (3). Then more stakeholders are invited to the roundtable for review in step (4). In step (5), regulators give final approval. And in step (6), this law is translated into code as part of the (ρro)Metheus Typed ρ-Calculus Covenant-Judgement system. This community-defined type or Covenant-Judgement can now be used to audit AI models and datasets for stakeholder compliance.

(ρro)Metheus replace the notion of "code is law" with "`type is law`." Here the subjects under legal audit are computer programs, not people.

The most important aspect of this Covenant-Judgement construction process is that there is a one-to-one correspondence between drafting and enforcing regulations, and designing types and type-checking code. This is (ρro)Metheus' interpretation of the adage "code is law," or alternatively the *Covenant-Type Correspondence*. This correspondence is outlined in table 1.

Law	Type
Assign Legal Status	Type Judgment or Covenant-Judgement
Drafting Legislation	Type Design
Core Constitution	Type Primitives
Legislation Amendment	Type Extension
Legislation Enforcement	Type Checking by Compiler
Upholding Covenant	Does Type check
Violating Covenant	Does not Type Check
Relationship before the Covenant	Data/Model Composition before the Compiler
Proving Action is Legal under Terms of Covenant	Proving Typed ρ-Calculus Maintain Logical Invariants

Table 1 The correspondences between legal covenants and type system in Typed ρ-Calculus is the theoretical underpinnings that give a *computational interpretation* to the notion that *compiling code is the same as applying the law*. Under this correspondence, when the (ρro)Metheus compiler type checks the Typed ρ-Calculus code, it is also auditing the AI agents and datasets for legal compliance.

1.2 Broad Stakeholder Perspective

The technical perspective may be sufficient for those with a background in both machine learning and programming language theory, however it is too abstract for the non-technical audience. This section pulls the lens back and recontextualize (ρro)Metheus within the technical and political milieu of the present moment. We answer the question: why is (ρro)Metheus necessary when a diverse set of tools exists to characterize the confidence, validity, and scope of statistical models and datasets?

Why (ρro)Metheus

The demand arises with the latest tranche of breakthroughs in AI around large language models (LLMs). Nominally LLMs are trained via regression on time series data, therefore they are not categorically different from i.e., deep convolution neural networks (CNNs) trained on non-time-series data, or a soft-object manipulations algorithm sampled via reinforcement learning. However the *user-behavior* that have spawned around LLMs applications differ significantly from earlier machine learning tools. Presently users are not only using LLMs as general purpose information retrieval systems, but they are also prompting them as "intelligent reasoning" engines. This user behavior spans across many settings:

- *Financial setting*: banks and insurance companies are using LLMs in high-valued contexts such as producing analyst reports to inform buy/sell decisions. They have never used CNNs in this way. Moreover, the prior generation of financial models used to price assets are either simple, i.e. linear regression on a few factors, or reasonably explainable models based on heat-diffusion equations. In contrast, LLMs are complex and opaque.
- *Medical setting*: doctors are using LLMs to summarize patients' past to inform an opinion, while patients are using LLMs to diagnose their illness. This has far reaching consequences extending past the hospital setting, including the pricing of insurance and other ancillary services.

- *Consumer setting*: people are treating LLMs as friends, teachers, and personal psychologists. This use case may appear the most trivial, however social media has already demonstrated its capacity to silo citizens into misinformation echo chambers and throw elections. Moreover, whereas the harm due to social media is bounded by the number of people who posts, LLM-based social media tools have no such upper bound. This presents a long-term challenge to the health of any society.

In summary, unlike prior deep learning models, LLMs function as "artificial people," or more to the point: "entry level employees." Now given the variety and sensitivity of LLMs' application domains, we assert LLM-based applications require stricter oversight, perhaps even legal oversight, from regulators beyond rudimentary internet guidelines.

1.2.1 The (ρro)Metheus Oracle Desiderata: a High Level Attempt

The (ρro)Metheus computational environment is a protected sandbox with strictly defined and enforced rules. In particular, the *type-system* of the (ρro)Metheus Typed ρ-Calculus is crafted to satisfy the following desiderata:

1. *Flexibility*: regulators from across jurisdictions may define laws governing the behavior of LLMs in a way that suit their local conditions. This is important since language models summarize the internet and with it: culture. There cannot be one set of universal "LLM-laws," as different cultures carry different standards of "good behavior." Therefore the defined boundary conditions must be different.
2. *Composability*: it would be prohibitively expensive, not to mention tedious, to define laws governing every invocation of an AI agent. Thus Typed ρ-Calculus language is defined so that regulators can specify simple rules, and when composed they form a rich system spanning the behavior of an AI agent.
3. *End-to-end oversight*: once the boundary conditions are defined, regulator-privilege extends deep into any AI-based system over the course of its application lifecycle. This means regulators may define high-level laws governing what is good AI behavior. Then the laws would propagate from data collation, to model-training, refinement, and finally inference as well as post-inference data-collection steps. If one conceptualizes an AI agent as a living being, then (ρro)Metheus regulates agent-behavior from cradle to grave.

In conclusion, the (ρro)Metheus Typed ρ-Calculus computational framework operationalizes the adage "code is law." In this case the subjects under legal audit are not people but computer programs. In more technical terms:

> The (ρro)Metheus Typed ρ-Calculus language is a new way of realizing *code as law*. It does so with a novel application of type systems common in statically-typed programming languages. Whereby AI agents and data constitutes the term universe, while the regulator legal framework defines the type universe. This type system, or legal framework, governs:
>
> - how AI learns;
> - once learned, how AI interacts with people, data, and each other.

1.2.2 The (ρro)Metheus Stakeholder Stack

The fact that (ρro)Metheus interacts with regulators at the sovereign level is a critical differentiator and unique go-to-market channel. Presently both financial and medical institutions are inundated with "AI offerings" from major firms and startups alike.

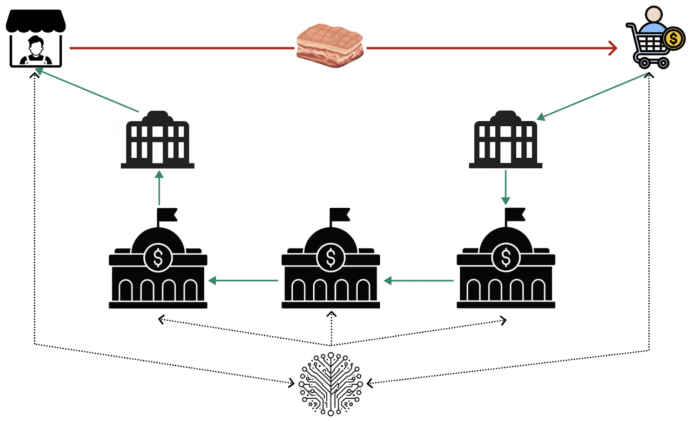


Figure 2 The (ρro)Metheus oracle applied to supply chain orchestration, whereby vendors sell foodstuff to buyers across national borders. The (ρro)Metheus oracle informs trade across stakeholders along this transnational network servicing: buyers, sellers, commercial/central banks, and insurers.

- The startups sell GPT wrappers: that is new way of selling the same old enterprise workflows. This game is stale and soon they shall be cannibalized.
- Major firms such as Microsoft and Google use "AI-agents" as cheese in the trap to sell what they have always sold: API calls and data egress. Ambitious managers use these projects to pad their promo-package. And once promoted, the projects are typically orphaned and whither away.

We escape this red ocean completely and appraise the stakeholder stack as follows:

- *Sovereign regulators*: (ρro)Metheus sell protected computational environments, whereby all code execution is compliant with respect to local laws and norms. The importance of security cannot be overstressed, especially as we enter a multipolar era whereby all things on the chessboard enter a state of motion. Since laws are public goods that rightly belong in the public domain, we engage with regulators through the nonprofit foundation FairCovenant. See the whitepaper at http://bit.ly/44EiQV0 for details on how we engage with regulators [9].
- *Enterprises*: (ρro)Metheus sells compliance as a service, so that the downsides of said institutions using AI are protected. This is critical: if AI is to make high valued decisions, then someone must be liable - (ρro)Metheus answers this question in a principled manner.
- *Enterprises Analysts*: (ρro)Metheus offers a high level interactive environment whereby analysts may query the data for causal relations. This dovetails how people are using LLMs anyways. However the statistical language describing this class of auto regressive models *cannot express causation*. The (ρro)Metheus Typed ρ-Calculus language fills this gap by expressing causality via a principled mathematical language.

The last point bears repetition. Recall Typed ρ-Calculus is not just for regulator-compliance, but also for *probabilistic causal reasoning*. We did not stress this aspect of (ρro)Metheus in the introduction, but it shall be specified in the main body shortly.

1.3 The Greater Metheus Ecosystem

Although (ρro)Metheus is a general purpose computational oracle, it is initially deployed for stakeholders along specific supply chains to support transnational trade organizations (see figure 2). Additionally, (ρro)Metheus will activate the larger open-source AI community by drawing both from

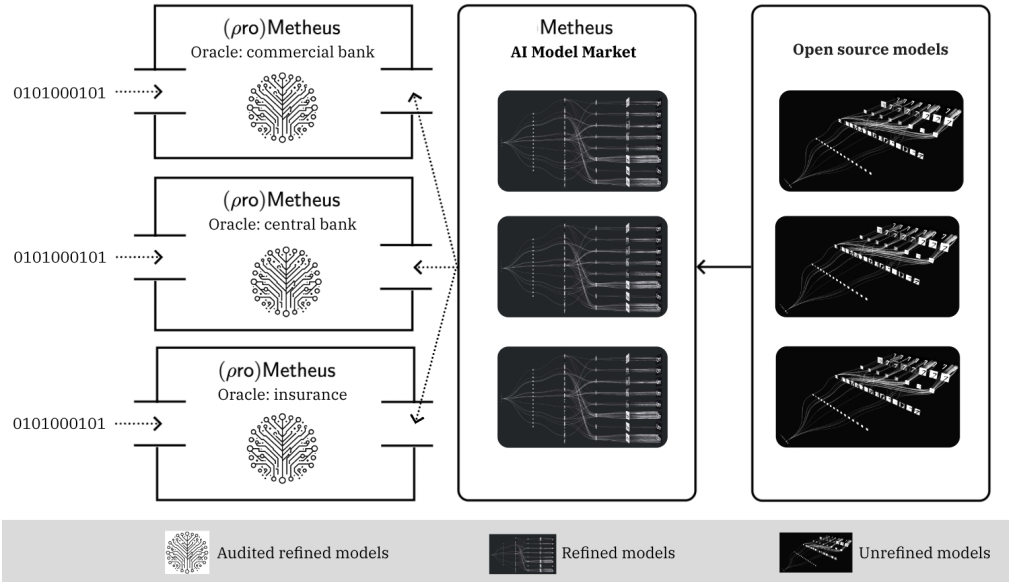


Figure 3 The (ρro)Metheus Oracle connects enterprises with the broader AI foundational model ecosystem by embedding refined AI models within a rigorous legal framework.

freely available models, as well as incentivizing the community to fine-tune or train new models. The process is outlined in figure 3:

- *left column*: there will be separate (ρro)Metheus oracle environment for each industry stakeholder. This is appropriate as each player has different standards of data validity, and AI model security. These separate requirements find congruence in the Typed ρ-Calculus type system.
- *center column*: (ρro)Metheus is connected to an open source AI model market, whereby foundational models are refined by distributed teams of ML engineers. [4]
- *right column*: this moment in technology is unique because of the gluttony of open source foundational models. We funnel this set of commodified assets into the privileged environment of (ρro)Metheus oracle.

Observe that (ρro)Metheus's moat is the regulator and stakeholder-defined laws of data validity and model security. That is say: **their wall is our moat.**

In (ρro)Metheus, the regulator's wall is our moat.

---

[4] See monograph at http://bit.ly/4lFn8Sf for details on AI model market [8].

2 The Covenant-Type Correspondence

This section underlines the principled basis of Typed ρ-Calculus, that is the correspondence between:

- a legal covenant,
- and types that mediate how program assets interface with each other.

This correspondence formalizes the notion that *compiling code is the same as applying the law*; or simply "code is law." Some definitions and examples are listed below, followed by more in depth discussion on the correspondence.

In (ρro)Metheus, code that compiles is also code that obeys the law. That is to say compliant computations run to completion, while illegal computations fail *before* execution. This is know as: `static guarantee of covenant compliance`.

2.1 Preliminary Definitions and Examples

▶ **Definition 2.** *[1] In law, a **legal covenant** is a binding agreement or promise, often included in a contract or deed, that obligates one party to perform or refrain from a specific action. It can be 1) positive covenant requiring an action, or 2) a negative covenant restricting an action.*

▶ **Example 3.** [2] In (Property Law), there are two types of restrictive covenants: affirmative and negative:

- An affirmative covenant obligates a person to act. For example, a covenant that requires the homeowner to keep the trees trimmed in the yard is an affirmative covenant.
- A negative covenant prohibits a person to act. For example, a negative covenant can forbid a homeowner from building a fence.

▶ **Definition 4.** *[5] A **Type System** is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute. Its theoretical basis lies in early 20th century study of logic, mathematics, and philosophy. In computational terms, programs equipped with types allows the software engineer and the compiler to reason about the run-time behavior of the program.* [5]

▶ **Example 5.** (`Int` and `String`) In *statically typed languages* such as `Haskell`, every data and function in the language is associated with a type. The type specifies the proper behavior the function, while the compiler ensures the program type-checks, or in more informal terms: "behaviorally correct" *before* the code runs. This removes entire class of errors from run time and ensure the code does not behave poorly as it executes. The following code block defines a function `strLen` counts the length of a `String`, in this case written as `[Char]`.

[5] A compiler is a software program that translates one programming language to another.

```
strLen :: [Char] -> Int
strLen [] = 0
strLen x:xs = 1 + strLen xs
```

Observe this function is typed so that it can only accept a string (in this case represented as list of **Char**acters), so that we have: `strLen "hi" = 2`. If instead the user passes a **Bool**ean value into the function with **StrLen** `true`, then the function `strLen` will fail outright since it is "contractually obligated" to only accept values of type `[Char]`.

2.2 The Correspondence

The type-covenant correspondence occurs in two phases: 1) the drafting of legislation or type system design, and 2) the enforcement of legislation or static type checking. [6] A summary of the similarities is found in table 2.

Law	Type
Assign Legal Status	Type Judgment
Drafting Legislation	Type Design
Core Constitution	Type Primitives
Legislation Amendment	Type Extension
Legislation Enforcement	Type Checking by Compiler
Upholding Covenant	Does Type check
Violating Covenant	Does not Type Check
Relationship before the Covenant	Data/Model Composition before the Compiler
Proving Action is Legal under Terms of Covenant	Proving Typed ρ-Calculus Maintain Logical Invariants

Table 2 The correspondences between legal covenants and type system in Typed ρ-Calculus is the theoretical underpinnings that give a *computational interpretation* to the notion that *compiling code is the same as applying the law*. Under this correspondence, when the Typed ρ-Calculus compiler type checks the Typed ρ-Calculus code, it is also auditing the code for legal compliance.

Now we expand table 2 as follows:

- *Legislation design*: the drafting of legislation corresponds to designing type system within Typed ρ-Calculus.
  1. In real life one finds a core constitution, with many extensions that flow from the core structure. Similarly within the (ρro)Metheus computational environment, one will find primitive type and semantics that define entities and their relations. This is followed by extensions defined by the non-profit foundation defined in [9].
  2. In particular, in (ρro)Metheus Typed ρ-Calculus, the correspondence between type and covenant arises in the negative direction. That is to say the Typed ρ-Calculus compiler prevent non-compliant AI agents from running.

[6] Normally we abhor the egregious anthropomorphization of machine learning in popular literature. But in this setting there is a fine parallelism of how human societies relate to the *legal machinery*, and how AI agents in (ρro)Metheus computational environment relate to the Typed ρ-Calculus type system. That is to say in both cases we focus on the rationalization or mechanization of relationships.

3. Finally just as covenants govern the behavior of people w.r.t each other before the state, in (ρro)Metheus the type system governs the *composition* of data with data, data with models, and model-to-model composition.

- *Legislation enforcement*: unlike covenants in the context of people, inside the (ρro)Metheus computational environment, AI agents and associated software cannot violate covenants at all. This is because as Typed ρ-Calculus mobilize computational assets such as data and AI agents, said assets are type-checked before they are used in regulated computations. The advantage of AI agents over people is that while people may violate covenants, digital covenants defined by Typed ρ-Calculus logic are absolute: no AI agents may break the type system provided they are bound to (ρro)Metheus. Therefore compliant computations run to completion, while illegal computations fail *before* execution. This property of the (ρro)Metheus environment is called *static guarantee of covenant fulfillment*.

3 Designing the (ρro)Metheus Covenant

3.1 Introduction

The (ρro)Metheus Typed ρ-Calculus language is an example of a domain-specific programming language or DSL. DSLs are common in industry, and span from simple scripting languages for ad-hoc tasks, to complex languages embedded into a far-reaching ecosystems. The primary example here is MatLab. A complete (ρro)Metheus computational environment is comparable to MatLab in complexity. However whereas MatLab is a tool with low level integration with the legal structure of academia, (ρro)Metheus is deeply embedded into the financial and legal aspects of transnational trade systems. This requires a broad surface area of engagements with legal, financial, and insurance entities servicing said trade routes.

Consequentially, the determination of core type or covenant primitives in the Typed ρ-Calculus DSL is part of a broader governance process of The FairCovenant Foundation outlined in [9]. In particular, FairCovenant collaborate closely with stakeholders to determine the proper regulatory requirements needed to deploy AI agents. These requirements are then encoded into the type system, or legal system of (ρro)Metheus computational environment. See figure 4 for how the process is denoted procedurally. [7].

Moreover, since the type system is shared by multiple stakeholders that span many jurisdictions, the type system itself exists as a *public good*, and therefore is financed as such. Since the exact design of the core type system is delegated to the FairCovenant Foundation, the rest of the paper will speak of types in the generic sense with symbol κ. Covenant type assignment is then written with:

$$\Gamma \models \texttt{agnt} : \kappa_1 \qquad \Gamma \models \texttt{data} : \kappa_2. \tag{1}$$

Expression 1 is read: under the *validation environment* Γ defined by The FairCovenant Foundation, the AI `agnt` satisfies covenant κ_1, and the `data` has covenant value κ_2. The symbol Γ is also referred to as "typing context" in programming language terms. See section 4.1 for more on type judgement.

3.2 Prior Art

Now we reify the type assignment of expression (1) with concrete illustrations. Over the past few years, there has been a growing call to better document machine learning models and the data they are

[7] Link to whitepaper: http://bit.ly/44EiQV0

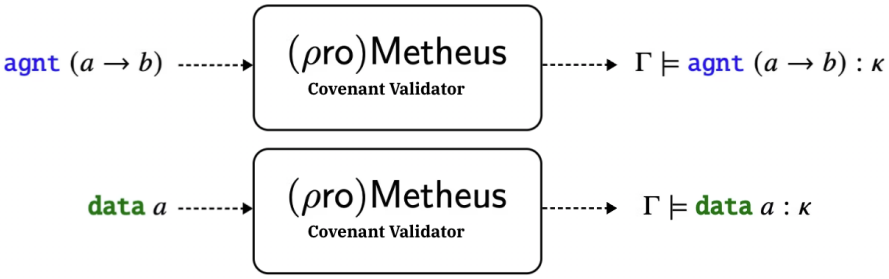


Figure 4 The (ρro)Metheus covenant-invalidator Γ ingests `data` or `agnt`, and assigns some Covenant-Judgement κ to each computational asset. The exact rules governing the content of the covenant κ is determined by The FairCovenant Foundation, so that once judged the `agnt`/`data` is deemed regulator-compliant. This κ now governs how said computational asset can interact with other assets in the (ρro)Metheus protected environment. Here compliant computations run to completion, while illegal computations fail *before* code execution. Thus illegal code cannot even run. This property is called: *static guarantee of covenant fulfillment*.

trained on. The next two examples represent preliminary attempts of assigning types to AI assets. The (ρro)Metheus Typed ρ-Calculus ecosystem will take the flavor of documentation presented in examples 6 and 7, and express them as types.

▶ **Example 6.** [3] (**Model Cards**) are documentations (or meta-data) that determines the application scope and provenance of machine learning models. This is an example of $\Gamma \models$ `agnt` $: \kappa$. From the abstract the authors state:

"Model cards are short documents accompanying trained machine learning models that provide benchmarked evaluation in a variety of conditions...that are relevant to the intended application domains. Model cards also disclose the context in which models are intended to be used, details of the performance evaluation procedures, and other relevant information... this framework can be used to document any trained machine learning model."

An example of documentation for a particular model is presented in figure 5.

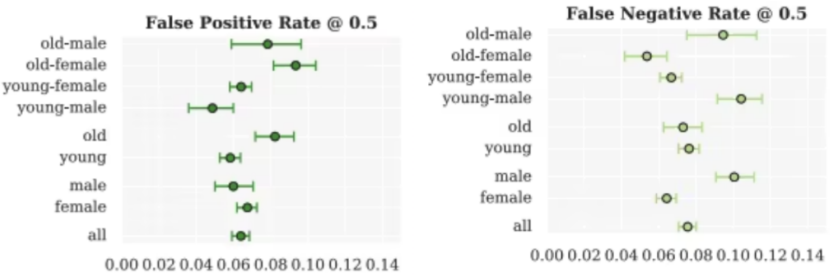


Figure 5 In this model for smiling detection in images reported in [3]. Here the false positive and false negative rates for each model is reported, broken down by demographic data. Documenting the performance envelope of a model is important if the model is to be deployed in sensitive areas, such as detecting the likelihood of a criminal from images. This likelihood could be used to make arrests, which presents a challenge as said estimations are often skewed by the exiting on criminals. In (ρro)Metheus Typed ρ-Calculus environment, the totality of a given model's metadata would be abstracted into a unique type **T**.

▶ **Example 7.** [4] (**Data Cards**). Similar to model cards, data cards document the provenance, validity, and applicability of data used to train machine learning models. This is an example of $\Gamma \models$ `data` $: \kappa$. From the abstract we have:

"we propose Data Cards for fostering transparent, purposeful and human-centered documentation

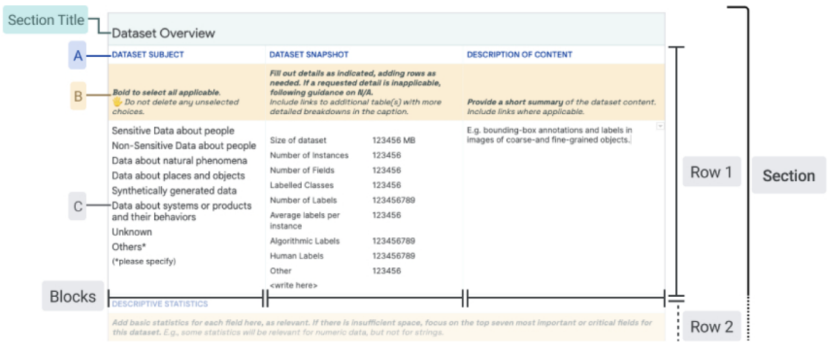


Figure 6 Similar to model cards, data cards document meta-information on the dataset used to train AI models. Since machine "learning" is data compression, this meta information is then appropriately inherited by any AI model trained on the data, along with any additional meta-information accumulated while the model is under training. Image from [4].

of datasets within the practical contexts of industry and research. Data Cards are structured summaries of essential facts about various aspects of ML datasets needed by stakeholders across a dataset's lifecycle for responsible AI development. These summaries provide explanations of processes and rationales that shape the data and consequently the models—such as upstream sources, data collection and annotation methods; training and evaluation methods, intended use; or decisions affecting model performance." An example data card format is show in figure 6.

The cited paper [3] on model cards is not just academically forward, the careers of the authors are also cautionary tales for those who push for regulation on AI agents without proper institutional support. In particular, the authors Margaret Mitchell and Timnit Gebru are of an "activist mindset." They would later author a different paper challenging the ability of deep learning models to think. No serious scholar in the field confuses deep learning model inference for "thinking." However the conduct of the authors nonetheless drew the ire of management at Google. They found themselves in the crosshair of a Distinguished Google Fellow, the two women were subsequently fired abruptly.

Thus example 6 is a case study of how *not* to push for closer scrutiny on AI models, especially when so many livelihoods, reputations, money, and egos are on the line. This is why (ρro)Metheus deploy AI regulation and legal framework through the nonprofit foundation FairCovenant. Only by working closely with regulators, insurers, and central banks in their domain of operation, can we externalize the necessary *legitimacy* to regulate artificial intelligence.

3.3 Formal Specification of the (ρro)Metheus Covenant

This section formally specifies the (ρro)Metheus Covenant as a privileged datatype in the Typed ρ-Calculus universe. Recall the details of the data validation criteria are designed and maintained by The FairCovenant Foundation. The format of said validation criteria is akin to a checklist. In the event whereby a criterion is real valued, then it can be discretized into regions. This region is then expressed as a binary tree, so that a real value falling within one of the zones correspond to a particular path down the tree. [8]

▶ **Definition 8.** *(An* `agnt` *or* `data` ***validity criteria*** *C) is a check-list of quality measures that the* `agnt` *or* `data` *must satisfy to partake in the* (ρro)Metheus *computational environment.*

[8] This formulation is elegant because the covenant itself is also structured as a binary tree, see definition 9.

The data and model cards of examples 7 and 6 are instances of validity criteria C.

▶ **Definition 9.** *(A* Covenant-Judgement κ*). For every set of* `agnt` *or* `data` *validity criteria C of length* $|C| = n$ *defined by The* FairCovenant *Foundation, so that each criteria in C is satisfied independently of another. Now let the* (ρro)Metheus *Oracle judges some* `agnt` *or* `data` *according to this criteria C, so that it generates a verdict* $\kappa_{2^n,j}$ *of length n, whereby:*

- $\kappa_{n,i}[j] = 1$ *if the* `agnt` *or* `data` *satisfies criterion j.*
- $\kappa_{n,i}[j] = 0$ *otherwise.*

This n-bit vector $\kappa_{n,i}$ *is a* Covenant-Judgement *w.r.t the criteria C. Moreover, the following three statements are equivalent:*

- `agnt` *or* `data` *has* Covenant-Judgement $\kappa_{n,i}$*;*
- `agnt` *or* `data` *witnesses* $\kappa_{n,i}$*.*
- *In formal notation:*

$$\Gamma \models \texttt{agnt} : \kappa_{n,i} \quad \text{or} \quad \Gamma \models \texttt{data} : \kappa_{n,i}.$$

We denote the set of all possible covenants generated by criteria C with the symbol $\mathbb{K}$. Observe that given a criteria set of length n, $\mathbb{K}$ is exactly the set of all paths down a binary tree of height n:

$$\mathbb{K}(C) = \{\kappa_{n,0}, \ldots, \kappa_{n,2^n-1}\}.$$

And each Covenant-Judgement $\kappa_{n,\mathrm{i}}$ is one path down this tree (see figure 7).

▶ **Example 10.** (**A covenant set** $\mathbb{K}(C)$) over three criteria is the set of paths down a binary tree of depth 3, and may be drawn:

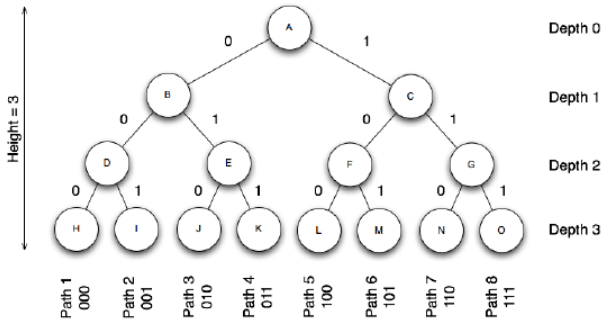


Figure 7 A binary tree of depth 3 induces $2^3 = 8$ possible 3-bit strings. Each string is an example of a covenant indexed by 3, written: $\kappa_{3,\mathrm{i}}$.

In this example, we may enumerate the possible Covenant-Judgements with:

$$\mathbb{K}(C) = \{\kappa_{3,0}, \kappa_{3,1}, \ldots, \kappa_{3,7}\} \text{ where}$$
$$\kappa_{3,0} = 000 \quad \kappa_{3,1} = 100 \quad \kappa_{3,2} = 010 \quad \kappa_{3,3} = 001 \quad \kappa_{3,4} = 110 \quad \kappa_{3,5} = 101 \quad \kappa_{3,6} = 011 \quad \kappa_{3,7} = 111,$$

so that each $\kappa_{3,x}$ is a particular Covenant-Judgement. For example if a particular `data` *a* satisfies the first and third criteria only, then `data` *a witnesses* 101. Its Covenant-Judgement is written:

$$\Gamma \models \texttt{data}\ a : \kappa_{3,5}, \quad \text{where } \kappa_{3,5} = 101.$$

▶ Remark 11. (**Simplified notation on** Covenant-Judgement). The notation $\kappa_{3,1}$ is cumbersome, so we prefer to write κ_1 instead, and leave the length of the bit-string 3 implicit.

Additionally, Typed ρ-Calculus must judge covenants by comparing them numerically, so that (ρro)Metheus may price covenant witnesses accordingly. Thus we will define a notion of distance and "less than." Hence we place an algebra on $\mathbb{K}$ as follows.

▶ **Definition 12.** *(**A covenant space** $\mathbf{K}$) with respect to some* FairCovenant *Foundation defined criteria C is a metric space equipped with a distance function. Written:*

$$\mathbf{K}(C) = (\mathbb{K}(C), \ ||\ ), \quad \text{where} \ \ ||\ : \mathbb{K} \times \mathbb{K} \to \mathbb{R}. \tag{2}$$

Where the distance function $||$ is defined by a suitable arithmetization of the Covenant-Judgement*s in $\mathbb{K}$. This arithmetization is defined by The* FairCovenant *Foundation, so that given arithmetization function* arith*, we have:*

$$\kappa_i \ || \ \kappa_j = \mathsf{arith}(\kappa_i) - \mathsf{arith}(\kappa_j). \tag{3}$$

From this point forward, when we say "covenant," we refer to the covenant space $\mathbf{K}(C)$, so that for every FairCovenant criteria of length n, there is an associated covenant space $\mathbf{K}(C)$. Now we consider the issue of Covenant-Judgement composition. This arises in settings when:

- Datasets are composed with other datasets, and they have different covenant criteria. For example, `data` a may be a set of images, while `data` b may be a set of image-aligned captions. They will satisfy different covenant criteria, however the combined dataset [9] will carry Covenant-Judgement information from the source datasets.
- Trained `agnt` that witnesses some Covenant-Judgement κ is refined on `data` -set of Covenant-Judgement κ'. Then the resultant model must have some type κ'', the content of this κ'' must be defined.
- The output of one `agnt` $(a \to b)$ is input some other `agnt` $(b \to c)$. If `agnt` $(a \to b)$ witness κ, while `agnt` $(b \to c)$ witness κ'. Then what is the resultant value c output by the model?

▶ **Definition 13.** *(**Covenant Composition**) between two covenant spaces $\mathbf{K}$ and $\mathbf{K}'$ generated by two different criteria C and C', is the Cartesian product of the two spaces. Written:*

$$\mathbf{K} \ \otimes \ \mathbf{K}' = \left(\left\{ \boldsymbol{\kappa}_m = (\kappa_i, \kappa_j) \ : \ \kappa_i \in \mathbf{K}, \kappa_j \in \mathbf{K}' \right\}, \ \ || \ \right). \tag{4}$$

Now we can define the distance $||$ between two product covenants by arithmetizing each of the elements in the vector with:

$$\boldsymbol{\kappa}_m = (\mathsf{arith}(\kappa_i), \mathsf{arith}(\kappa_j)). \tag{5}$$

Thereby interpreting this κ_m as a real-valued vector. Then the distance function is defined as the dot product between the two vectors:

$$\kappa_m \ || \ \boldsymbol{\kappa}_n = \boldsymbol{\kappa}_m \cdot \boldsymbol{\kappa}_n. \tag{6}$$

Some further observations follow:

[9] Sometimes called a multi-modal dataset.

- Covenants are closed under composition. That is to say the product of covenants are a covenant themselves.
- Covenant composition is not necessarily commutative. That is to say we should not expect this to be true: $(\kappa_i, \kappa_j) = (\kappa_j, \kappa_i)$.
- We may introduce an identity covenant for every Foundation-defined covenant criteria, by defining this identity as the all-ones vector of suitable length. In vector space language, this corresponds to lifting some sub-space into the affine-space.

One final definition will round out the chapter.

▶ **Definition 14.** *(**The** (ρro)Metheus **Covenant Universe** $\mathfrak{K}$) is generated as follows:*

1. *Given the set of all possible criteria $C, \ldots$ The* FairCovenant *Foundation defines, we generate the set of all possible covenant spaces with:*

$$\mathbb{S} = \left\{ \mathbf{K}(C_i), \ldots \;\middle|\; C_i \in \{C_1, \ldots\} \right\}.$$

2. *Now construct the sigma-algebra over $\mathbb{S}$ with $\sigma(\mathbb{S})$.*
3. *And complete each subset $\mathbb{S}$ in $\sigma(\mathbb{S})$ under $\otimes$ with:*

$$\mathfrak{K} = \left\{ \otimes_{\mathbf{K}(C_i) \in \Sigma} \mathbf{K}(C_i) \;\middle|\; \Sigma \in \mathbb{S} \right\}. \tag{7}$$

In the remainder of the paper, we define what it means to assign every program in Typed ρ-Calculus some Covenant-Judgement in $\mathfrak{K}$. This concludes the discussion on the formal specification of Covenant-Judgements. In summary:

1. Covenants are specified by The FairCovenant Foundation, it is a list of criterion that `agnt` and **`data`** must satisfy to partake in the (ρro)Metheus computational enviornment.
2. A particular instance of a Covenant-Judgement is just a bit string, which when appropriately arithmetized, becomes a real number that can be subtracted.
3. When a computational asset such as `agnt` witnesses some Covenant-Judgement κ, we assign the `agnt` with this Covenant-Judgement with the notation: $\Gamma \models$ `agnt` $: \kappa$.
4. When computational assets compose, the Covenant-Judgements also compose accordingly. Composition occurs when:
   - **`data`** compose with other **`data`** to augment information content.
   - `agnt` compose with **`data`** during training or inference.
   - `agnt` compose with other `agnt` when the output of one `agnt` is used as input into another `agnt`.

   Composition allows us to build complex Covenant-Judgements, or *legal obligations* from atomic Covenant-Judgement definitions. This greatly augment (ρro)Metheus' ability to audit **`data`** and `agnt`s. The FairCovenant Foundation member's time is valuable, composition allows them to define primitive obligations that code must satisfy, so that the these covenant-primitives *span* a set of acceptable AI behavior in the (ρro)Metheus universe.

4 Formal Specification of Typed ρ-Calculus Layer I

The rest of the paper recursively descends downwards, so as to define Typed ρ-Calculus for covenant-bound probabilistic causal reasoning. The language is introduced by layers as follows:

1. We begin at the highest level of abstraction, and define how models and data interact with each other in the aggregate. In particular, we are sensitive to how stakeholders in the FairCovenant Foundation defines how AI assets interact with each other.
2. Next we descend down one level, and express how Typed ρ-Calculus can be used to build a probable causal graph given information drawn from raw data and information summarized by LLM models. We then show how the type system interacts with the causal graph to properly assign legal culpability as Typed ρ-Calculus computes probable cause of an event.
3. Finally, we propose future directions whereby the type system may find coverage, for example in the computational graph of deep learning models in i.e. TensorFlow.

In this chapter, we first introduces the basic technical knowledge needed to understand Typed ρ-Calculus and Loare-Logic, and then specify layer one in point (1) above.

4.1 Technical Background

This section provides the necessary background from the field of programming languages and formal verification to understand Typed ρ-Calculus used to compute causal relations from data, and the (ρro)Metheus Loare-Logic used to generate proofs of Covenant-Compliance.

Language Syntax and Typing Relations

Given a typed programming language, its *typing relations* is an inference rule that describes how the type system assigns a type to a syntactic construction.

- These rules are applied by the type system to determine if a program is well-typed and what type the expressions have.
- If the program is well-typed, then it should satisfy certain properties such as all programs can either step forward or terminate with a value.
- Types restrict the universe of allowed programs so that all well-typed programs is "well behaved" according to the designer's desiderata [5].

An example follows to demonstrate the interplay between the syntax of a language and its typing relations.

▶ **Example 15.** [6](**Simply-Typed Lambda Calculus**). Given a language with syntax:

expressions	$\mathsf{e} := \mathsf{x} \mid \lambda \mathsf{x} : \tau.\mathsf{e} \mid \mathsf{e}_1\mathsf{e}_2 \mid \mathsf{n} \mid \mathsf{e}_1 + \mathsf{e}_2 \mid ()$
values	$\mathsf{n} := \lambda \mathsf{x} : \tau.\mathsf{e} \mid \mathsf{n} \mid ()$
types	$\tau := \texttt{Int} \mid \texttt{Unit} \mid \tau_1 \to \tau_2.$

We introduce a relation or judgment over typing contexts Γ, expressions e, and types τ. The judgment:

$$\Gamma \models \mathsf{e} : \tau,$$

is read as "expression e has type τ in context Γ." Next we write the type judgement over all expressions:

$$\frac{}{\Gamma \models n : \texttt{Int}} \qquad \frac{}{\Gamma \models () : \texttt{Unit}} \qquad \frac{\Gamma(x) = \tau}{\Gamma \models x : \tau}$$

$$\frac{\Gamma \models e_1 : \texttt{Int}, \Gamma \models e_2 : \texttt{Int}}{\Gamma \models e_1 + e_2 : \texttt{Int}} \qquad \frac{\Gamma, x : \tau \models e : \tau'}{\Gamma \models \lambda x : \tau . e : \tau \to \tau'} \qquad \frac{\Gamma \models e_1 : \tau \to \tau' \; \Gamma \models e_2 : \tau}{\Gamma \models e_1 e_2 : \tau'}$$

Observe this set of typing rules are applied to each expression and determines how the Lambda Calculus expressions can be typed. Furthermore, line two shows the inductive typing rules whereby complex expressions featuring function application and addition are also typed.

The (ρro)Metheus Loare-Logic replicates this design pattern, by applying FairCovenant-defined Covenant-Judgement to the set of Typed ρ-Calculus expressions defined in eqn (8).

Hoare Logic

Hoare logic is a formal system with a set of logical rules for reasoning rigorously about the correctness of computer programs. Hoare-style verification is based on the idea of a specification as a contract between the implementation of a function its clients:

- The specification consists of the precondition and a postcondition.
- The precondition is a predicate describing the condition of the code or functions relies on for correct operation. The client must fulfill this condition.
- The post condition is a predicate describing the condition of the function establishes after correct running; the client can rely on this condition being true after the call to the function.

The goal of Hoare-style verification is thus to statically prove that, given a pre-condition, a particular post-condition will hold after a block of code executes. We do so by generating a logical formula known as a *verification condition*, constructed so that if true, we know that the program behaves as expected. From a syntactic standpoint, Hoare logic is presented to the programmer as *Hoare triples*.

▶ **Definition 16.** *[6] (**A Hoare Triple**) is a claim about the state before and after executing a command. The standard notation is:*

$$\{\{P\}\}\ \varphi\ \{\{Q\}\}.$$

Meaning:

- *If command φ begins execution in a state satisfying assertion P,*
- *and if φ eventually terminates in some final state,*
- *then that final state will satisfy the assertion Q.*

Assertion P is called the precondition of the triple, and Q is the postcondition. [10]

▶ **Example 17.** [6] (**Hoare Triple**) The following:

$$\{\{\mathbf{x} = 0\}\}\ \mathbf{x} := \mathbf{x} + 1\ \{\{\mathbf{x} = 1\}\},$$

is a valid Hoare triple, stating that command $\mathbf{x} := \mathbf{x} + 1$ will transform a state in which $\mathbf{x} = 1$ to a state in which $\mathbf{x} = 1$.

[10] We use double brackets for pre and post conditions, following [6].

▶ **Example 18.** (**Hoare Logic**) applied to a programming language would proceed as follows. Given a simple language:

arithmetic expressions	$\mathsf{e} := \mathsf{x} \mid \mathsf{n} \mid \mathsf{e}_1 + \mathsf{e}_2 \mid \mathsf{e}_1 \times \mathsf{e}_2$
boolean expressions	$\mathsf{b} := \texttt{true} \mid \texttt{false} \mid \mathsf{e}_1 < \mathsf{e}_2$
commands	$\mathsf{c} := \texttt{skip} \mid \mathsf{x} := \mathsf{e} \mid \mathsf{c}_1 ; \mathsf{c}_2.$

Its set of Hoare triples are:

$$\frac{}{\{\{P\}\}\ \texttt{skip}\ \{\{Q\}\}} \qquad \frac{}{\{\{P[\mathsf{e}/\mathsf{x}]\}\}\ \mathsf{x} := \mathsf{e}\ \{\{P\}\}}$$

$$\frac{\{\{P\}\}\ \mathsf{e}_1\ \{\{R\}\},\ \{\{R\}\}\ \mathsf{e}_2\ \{\{Q\}\}}{\{\{P\}\}\ \mathsf{e}_1 ; \mathsf{e}_2\ \{\{Q\}\}}$$

Now given any program written in the language above, we can use these Hoare rules to verify the program satisfy certain conditions laid out in Q and P.

In the context of (ρro)Metheus:

- the precondition P defines the quality of data and statistical models that are input into the analysis;
- the command φ is the Typed ρ-Calculus computation that determines the output;
- while the post condition Q is quality of output that the analyst-written (ρro)Metheus program is contractually obligated to output.

The terms of contracts are determined by FairCovenant in accordance to all stake-holder demands. The core feature of Loare-Logic is to guarantee that Typed ρ-Calculus programs that cannot satisfy contractually-obligated standards of quality fail at compile time, *before* the program is run. This *static guarantee* of quality satisfies the dual mandate of:

- ensuring the safety and validity of model outputs;
- and saving money as full analytic runs and/or training runs are expensive.

4.2 Typed ρ-Calculus **Expressions**

At the top level, Typed ρ-Calculus formally specify the notion of atomic computational assets, and computation over said assets in the (ρro)Metheus universe. In particular, we define the following three type of asset composition:

- *(**data** -**data** composition)*: arises when data scientists augment datasets by combining it with other sources of data. Here the pressure point is to ensure the dataset does not degrade in quality under augmentation.
- *(**data** -**agnt** composition)*: arises when an **agnt**s consume data either in training, or in inference.
- *(**agnt**-**agnt** composition)*: arises when one **agnt** calls on another **agnt** to complete a sub-task. Now since the output of **agnt** could also be interpreted as **data** , this composition may be decomposed as a sequence of **data** -**agnt** compositions.

▶ Remark 19. We present the salient aspect of Typed ρ-Calculus while keeping the syntactical rigor to a minimum. This elevates the core features of the (ρro)Metheus computational environment, without inundating the readers with implementation details of a fully featured domain-specific programming language.

4.2.1 Elementary Typed ρ-Calculus Expressions

$$\begin{aligned} &\textit{atomic assets:} && \texttt{asset}\ a\ b := \texttt{data}\ a \mid \texttt{agnt}\ (a \to b) \\ &\textit{elementary functions:} && \texttt{fns} := \texttt{train} \mid \texttt{infer} \\ &\textit{covenant judgments:} && \kappa \in \mathfrak{K} \end{aligned} \tag{8}$$

In the rest of this paper, we will use φ to refer to *atomic assets* and *elementary functions* in Typed ρ-Calculus, as well as inductively defined expressions built from said code. Some more commentary follows:

- (*line 1*): in Typed ρ-Calculus, the atomic `asset` are either datasets `data` or AI agents `agnt`s. The expressions are parameterized by generic types a and b, which could be i.e. `String` or `Int` types. Note these parametrized types are not the same as the Covenant-Judgement, which specifies quality of data or model that has been audited by The (ρro)Metheus Oracle.
- (*line 2*): the elementary functions are `train` which takes an `agnt` $(a \to b)$ and (`data` a, `data` b) as input and outputs a trained `agnt` $(a \to b)$. Or `infer` which takes in `agnt` $(a \to b)$ as input along with query `data` a, and output inferred response `data` b.
- (*line 3*): the Covenant-Judgement of Typed ρ-Calculus is the entire covenant judgement universe as defined in def (14).

▶ **Example 20.** (**Document Data**) A set of raw conversation transcripts has type '`data String`'.

▶ **Example 21.** (**Dialogue Systems**) An automated dialogue system or chat-bot that accepts queries in string, and respond with raw text has type '`agnt` (`String` → `String`)'.

4.2.2 Group Composition over `asset` Expressions

Now given a set of words, we can "place a grammar" over this set, so that the words may compose into sentences. Similarly, if one conceptualizes `asset` as elementary programs or words, then one can "place an algebra" on `asset` so that they compose to form larger programs. The algebra of choice in Typed ρ-Calculus is the `Semigroup`.

▶ **Definition 22.** *(`Semigroup`) A semigroup is an ordered pair* $(\mathbb{S}, \times)$ *such that* $\mathbb{S}$ *is a non-empty set, and* $\times$ *is an associative binary operation on* $\mathbb{S}$.

▶ Remark 23. Addition and multiplication over integers are both examples of semigroups. In the case of integers, there is also an identity element associated with summation in 0. So that addition forms a *group* in (`Int`,+,0), and similarly for (`Int`,×,1) in the case of multiplication. In the case of `asset` , the identity asset is not necessarily defined, hence the restriction to semigroup.

The next definition specifies `asset` composition by placing the Semigroup algebra on `data` :

$$\begin{aligned} &\texttt{Class Semigroup}\ (\texttt{data}\ a)\ \text{ where} \\ &\qquad \diamond :: \texttt{data}\ a \times \texttt{data}\ a \to \texttt{data}\ a \end{aligned} \tag{9}$$

▶ **Example 24.** (`data` **Composition**). Given two block of raw texts s_1 and s_2, each of type `data String`, we can concatenate the two blocks with $\diamond$ to form a new data set of type `data String`. Written:

$s_1 \diamond s_2 ::$ `data String`.

Similarly, we place Semigroup operation on agnt as follows:

$$\textbf{Class Semigroup } (\texttt{agnt}\ (\cdot \to \cdot)) \text{ where} \\ \rightarrowtail :: \texttt{agnt}\ (a \to b) \times \texttt{agnt}\ (b \to c) \to \texttt{agnt}\ (a \to c) \tag{10}$$

▶ **Example 25.** (**agnt Composition**). Given an AI agent agnt (String → String) that consumes text and output text (i.e. a chat-bot), and another AI agent agnt (String → Image)′ that consumes text and generates images. Then we can compose them with:

$$\left(\texttt{agnt}\ (\texttt{String} \to \texttt{String}) \rightarrowtail \texttt{agnt}\ (\texttt{String} \to \texttt{Image})'\right) :: \texttt{agnt}\ (\texttt{String} \to \texttt{Image}).$$

And now we have a new AI agent that outputs images when prompted.

4.2.3 Elementary Functions over asset Expressions

Now we define two functions that operate on asset . These correspond to elementary operations in the (ρro)Metheus machine learning ecosystem.

$$\texttt{train} :: \texttt{data}\ a \times \texttt{data}\ b \times \texttt{agnt}\ (a \to b) \to \texttt{agnt}\ (a \to b) \tag{11}$$
$$\texttt{infer} :: \texttt{data}\ a \times \texttt{agnt}\ (a \to b) \to \texttt{data}\ b \tag{12}$$

Commentary below:

- Function (11) takes in some dataset (data a, data b), and machine learning model agnt $(a \to b)$, and trains the model to output a trained ML function agnt $(a \to b)$. This signature is written for training discriminative models with labeled input pairs. In the case of generative models, data b is simply passed into the train function as the empty set.
- Function (12) takes a model agnt $(a \to b)$, and accepts an input (or prompt in the case of LLMs) data a, infers its value and outputs a prediction or response of type data b.

This ends the discussion on Typed ρ-Calculus top level expression. In summary:

- The atomic asset of the (ρro)Metheus computational environment are data and agnt. They must be audited according to FairCovenant-defined rules before use client-side.
- The elementary machine learning functions that operate over said asset are train and infer, they take data and agnt as inputs, and output the appropriate asset in response.
- The composition operators $\rightarrowtail$ and $\diamond$ compose agnt and data respectively. These semigroup functions, along with train and infer, allow the analyst to build complex Typed ρ-Calculus programs from atomic Typed ρ-Calculus asset .

In the next section, we introduce a novel concept termed Loare-Logic. It is the workhorse logical system used by the (ρro)Metheus compiler to validate Typed ρ-Calculus programs for Covenant-Compliance, before any AI-related computation is run.

4.3 Loare-Logic: **Axiomatic Semantics with Legalistic-Hoare Logic**

This section lays out the mathematical tools needed to enforce Covenant-Compliance of Typed ρ-Calculus expressions within the (ρro)Metheus computational environment. Section 4.3.1 defines Loare-Logic syntactically as a set of Covenant-Judgement rules, as well the mechanisms of how pre and post-conditions are enforced to ensure Covenant-Compliance. Then we enumerate all the rules in the subsequent section:

- Section 4.3.2 assigns Covenant-Judgement onto each atomic `asset` . This is equivalent to designating legal status to each entity in (ρro)Metheus, thereby reifying the type-covenant correspondence asserted in chapter 2.
- Sections 4.3.3 and 4.3.4 map `asset` composition onto Covenant-Judgement composition as defined by the product operation of definition (13). This ensures `asset` composed under $\diamond$ and $\rightarrowtail$ also witness the appropriate Covenant-Judgements, and are therefore Covenant-Compliant to the appropriate degree.
- Section 4.3.5 defines how Covenant-Judgement maps over the elementary function `infer` and `train`. That is to say: how does `infer`ence and `train`ing affect the output values' Covenant-Judgements.
- Section 4.3.6 provide some example proofs, demonstrating how Loare-Logic enforces Covenant-Compliance in complex Typed ρ-Calculus programs built inductively from `asset` primitives.

4.3.1 Loare-Logic **Introduction**

Loare-Logic is a formal system used by the (ρro)Metheus compiler to analyze the quality of data and model assets used in Typed ρ-Calculus programs. Loare-Logic ensures the analytic output of said programs are covenant compliant according to FairCovenant-defined rules.

▶ **Definition 26.** *(*Loare-Logic*) is a two-layered proof system that either generates a proof stating some* Typed ρ-Calculus *is covenant compliant, or terminates program execution for covenant violations. The system is layered as follows:*

- *A set of atomic and inductively defined rules assigning* Covenant-Judgement *to each expression in the* Typed ρ-Calculus *universe. Specifically, the following set of rules:*

$$\begin{aligned} \textit{rules} := \{&\texttt{data-L}, \texttt{agent-L}, \texttt{compa-L}, \texttt{compb-L}, \texttt{seql-L}, \texttt{seqr-L}, \texttt{seqd-L}, \\ &\texttt{train-L}, \texttt{infer-L}\}, \end{aligned}$$

  assign each user-defined expression in Typed ρ-Calculus *onto some covenant κ in the covenant universe $\mathfrak{K}$. In the case where the expression φ is some atomic `asset` , then the rule is written:*

$$\textit{rule:} \quad \frac{}{\Gamma \models \varphi : \kappa}$$

  If the expression is inductively defined using Semigroups operations $\diamond$ or $\rightarrowtail$, and/or using elementary functions such as `train` and `infer`, then the rule is written:

$$\textit{rule:} \quad \frac{\Gamma \models \varphi_1 : \kappa_1, \Gamma \models \varphi_2 : \kappa_2}{\Gamma \models \varphi_1\varphi_2 : \kappa_3}$$

  Stating that if expression φ_1 witness Covenant-Judgement κ_1 *while φ_2 witness κ_2, then their combined expression $\varphi_1\varphi_2$ has* Covenant-Judgement κ_3. *The exact rules to induce the judgement κ_3 is defined by* (ρro)Metheus *compiler in conjunction with* FairCovenant *Foundation regulations.*

- *Building upon the rule primitives are a set of contractual obligations decorating* Typed ρ-Calculus *programs. They govern the execution of* Typed ρ-Calculus *programs, guaranteeing that the pre and postcondition of said programs are covenant compliant. Syntactically, this is written:*

$$\{\{\Gamma \models \mathsf{x}_1 : \kappa_1, \ldots, \mathsf{x}_n : \kappa_n\}\}\ \varphi(\mathsf{x}_1, \ldots, \mathsf{x}_n)\ \{\{\Gamma \models \mathsf{y} : \kappa' \geq \kappa''\}\},$$
$$\text{where } \mathsf{y} = \varphi(\mathsf{x}_1, \ldots, \mathsf{x}_n), \quad \text{with } \kappa_1 \in \mathbf{K}_1, \ldots \kappa_n \in \mathbf{K}_n, \quad \text{and } \kappa', \kappa'' \in \mathbf{K}'.$$

  Stating that:

  - *given some input assets* $\mathsf{x}_1, \ldots, \mathsf{x}_n$ *of* Covenant-Judgement $\kappa_1, \ldots, \kappa_n$*;*
  - *and* Typed ρ-Calculus *expression* φ *that transforms the inputs onto output* $\mathsf{y} = \varphi(\mathsf{x}_1, \ldots, \mathsf{x}_n)$*;*
  - *the output* y *must witness some* Covenant-Judgement $\kappa' \geq \kappa''$.

  The threshold value of data quality κ'' *is called* ***the boundary condition on the output of computation*** *as enforced by* Loare-Logic*. It ensures the resultant prediction or response from the* Typed ρ-Calculus *code* φ *meets the minimum standards of quality as defined by the* FairCovenant *Foundation. The pre and postconditions defined in the brackets* $\{\{\cdot\}\}$ *will automatically "decorate" programs written by analysts in some specific instance of the* (ρro)Metheus *computational environment.*

Observe how Loare-Logic combines the syntax of type relations in example (15), and that of Hoare logic in definition (16) to express contractual enforcement of Typed ρ-Calculus programs w.r.t pre-agreed Covenant-Judgement. Now we specify the content of each *rule* enumerated in the set above, and provide examples of the Loare-Logic triple in action.

4.3.2 Elementary Covenant-Judgement over Computational Assets

The judgment of individual `data` and `agnt asset` are done by The (ρro)Metheus Oracle according to FairCovenant Foundation defined rules. In this case we simply assume they are assigned some appropriately-defined Covenant-Judgement κ as follows:

$$\texttt{data-L}\ \frac{}{\Gamma \models \texttt{data}\ a : \kappa} \qquad \texttt{agent-L}\ \frac{}{\Gamma \models \texttt{agnt}\ (a \to b) : \kappa}$$

4.3.3 Semantics of `data` Composition w.r.t Covenant-Judgement

The following set of expressions takes the semigroup composition over `data` and `agnt` defined in expressions (9) and (10), and map them onto Covenant-Judgement composition as defined in expression (4).

$$\texttt{compa-L}\ \frac{\Gamma \models \texttt{data}\ a : \kappa_1, \Gamma \models \texttt{data}\ a : \kappa_2, \kappa_1 \leq \kappa_2, \kappa_1, \kappa_2 \in \mathbf{K}}{\Gamma \models \texttt{data}\ a\ \diamond\ \texttt{data}\ a : \kappa_1}$$

$$\texttt{compb-L}\ \frac{\Gamma \models \texttt{data}\ a : \kappa, \Gamma \models \texttt{data}\ b : \kappa', \kappa \in \mathbf{K}, \kappa' \in \mathbf{K}'}{\Gamma \models \texttt{data}\ a\ \diamond\ \texttt{data}\ b : \kappa\ \otimes\ \kappa'}$$

- In `compa-L`, we are given some `data` a of Covenant-Judgement κ_1, and `data` a judged to be κ_2, so that the first judgement κ_1 is less "valuable" than the second. Then in this case, the entire dataset default to the less valuable judgment κ_1.

- In `compb-L`, there is some **data** a and **data** b, observe they are of different parameterized types. For example we may have **data String** and **data Image**, then their concatenation is the multi-modal dataset **data String** $\diamond$ **data Image**. In this case their covenants are drawn from different spaces, that is to say $\kappa \in \mathbf{K}$ and $\kappa' \in \mathbf{K}'$. Naturally the multi-modal dataset is in the product space $\kappa \otimes \kappa' \in \mathbf{K} \times \mathbf{K}'$.

▶ **Example 27.** (**compa-L and compb-L**) Suppose the following computational assets are loaded in the (ρro)Metheus environment:

$$\Gamma \models \textbf{data Int} : \kappa_1, \quad \Gamma \models \textbf{data Int}' : \kappa_2, \quad \Gamma \models \textbf{data Image} : \kappa,$$
$$\text{with } \kappa_1 < \kappa_2, \kappa_1, \kappa_2 \in \mathbf{K},\ \kappa \in \mathbf{K}'.$$

Then if the analyst build composite datasets, they will have the following inductively-defined Covenant-Judgement:

$$\Gamma \models \textbf{data Int} \diamond \textbf{data Int}' : \kappa_1, \qquad \Gamma \models \textbf{data Int} \diamond \textbf{data Image} : \kappa_1 \otimes \kappa.$$

Note that since **data Int** and **data Int**′ witness different elements of the same Covenant-Judgement space, the lower-valued Covenant-Judgement takes precedent. Where as **data Int** and **data Image** are drawn from different Covenant-Judgement spaces, thus the composite dataset rests in the product space of Covenant-Judgements or $\mathbf{K} \times \mathbf{K}'$.

▶ Remark 28. (**Notation on type and** Covenant-Judgement) In the code given in example 27, we use the notation $\Gamma \models$ **data** $a : \kappa$ to signify a computational `asset` of type **data** a has Covenant-Judgement κ. Technically, it is more complete to write, i.e.:

$$\Gamma \models \mathsf{x} :: \textbf{data}\ a : \kappa_1,$$

to signify that some dataset of value x of type **data** a has Covenant-Judgement κ_1. However this introduces another symbol x, which is cumbersome. Hence in all of our examples we leave the variable name x implicit.

4.3.4 Semantics of `agnt` Composition w.r.t Covenant-Judgement

Now we consider how sequencing over `agnt` by $\rightarrowtail$ affect Covenant-Judgement.

$$\texttt{seql-L}\ \frac{\Gamma \models \texttt{agnt}\ (a \to a) : \kappa_1, \Gamma \models \texttt{agnt}\ (a \to a)' : \kappa_2, \kappa_1 \leq \kappa_2, \kappa_1, \kappa_2 \in \mathbf{K}}{\Gamma \models \texttt{agnt}\ (a \to a)'' : \kappa_1}$$

$$\texttt{seqr-L}\ \frac{\Gamma \models \texttt{agnt}\ (a \to a) : \kappa_2, \Gamma \models \texttt{agnt}\ (a \to a)' : \kappa_1, \kappa_1 \leq \kappa_2, \kappa_1, \kappa_2 \in \mathbf{K}}{\Gamma \models \texttt{agnt}\ (a \to a)'' : \kappa_1}$$

$$\texttt{seqd-L}\ \frac{\Gamma \models \texttt{agnt}\ (a \to b) : \kappa_1, \Gamma \models \texttt{agnt}\ (b \to c) : \kappa_2, \kappa_1 \in \mathbf{K}, \kappa_2 \in \mathbf{K}'}{\Gamma \models \texttt{agnt}\ (a \to c) :\ \exists \kappa_3 \in \mathbf{K}'' \text{ s.t. s.t. } \kappa_3 = \texttt{proj}(\kappa_1 \otimes \kappa_2)}$$

$$\text{where } \texttt{proj}(\kappa_1 \otimes \kappa_2) := \text{arith}(\mathbf{K}'') \cdot \sqrt{\left(\frac{\text{arith}(\kappa_1)}{\text{arith}(\mathbf{K})}\right)^2 + \left(\frac{\text{arith}(\kappa_2)}{\text{arith}(\mathbf{K}')}\right)^2}.$$

- In `seql-L` and `seqr-L`, one of the `agnt` witnesses a less valuable Covenant-Judgement than the other. In both cases the combined `agnt` $(a \to a)''$ defaults to the less valuable judgement κ_2.

- In `seqd-L`, the `agnt` are parametrized by different types: `agnt` $(a \to b)$ v.s. `agnt` $(b \to c)$. Consequentially, their Covenant-Judgement are drawn from different covenant spaces, and the composed `agnt` $(a \to c)$ is drawn from a third covenant space $\mathbf{K}''$. Thus, we have to search for the most suitable Covenant-Judgement $\kappa_3 \in \mathbf{K}''$ that we *expect* `agnt` $(a \to c)$ to witness. We use the following procedure:
  1. We interpret the Covenant-Judgement `agnt` $(a \to b)$ and `agnt` $(b \to c)$ as two sides of a triangle, so that the Covenant-Judgement of `agnt` $(a \to c)$ is the hypotenuse. The objective is now to find the length of this hypotenuse.
  2. Next we arithmetize κ_1 and κ_2 using the 'arith' function, and determine the length of the hypotenuse with the Pythagorean theorem.
  3. This arithmetized value of κ_3 is then un-arithmetized onto some $\kappa_3 \in \mathbf{K}''$.

▶ **Example 29.** (**seqd-L Composition**). Suppose we have this set of (ρro)Metheus computational `asset` s, and their Covenant-Judgement:

$$\Gamma \models \texttt{agnt}\ (a \to b) : \kappa_2 \in \mathbf{K},\ \text{s.t. arith}(\kappa_2) = 3,\ \text{and arith}(\mathbf{K}) = 10,$$
$$\Gamma \models \texttt{agnt}\ (b \to c) : \kappa_6' \in \mathbf{K}',\ \text{s.t. arith}(\kappa_6) = 7,\ \text{and arith}(\mathbf{K}') = 20,$$
$$\text{and arith}(\mathbf{K}'') = 12,\ \text{with } \mathbf{K}'' = \{\kappa_0'', \kappa_1'', \ldots, \kappa_{11}''\}.$$

Then we know that under $\rightarrowtail$ composition, the Covenant-Judgement of the combined value is:

$$\texttt{proj}(\kappa_2 \otimes \kappa_6') := \text{arith}(\mathbf{K}'') \times \sqrt{\left(\frac{3}{10}\right)^2 + \left(\frac{7}{20}\right)^2} = 5.53,$$
$$\text{let } \texttt{agnt}\ (a \to c) := \texttt{agnt}\ (a \to b) \rightarrowtail \texttt{agnt}\ (b \to c),$$
$$\Gamma \models \texttt{agnt}\ (a \to c) : \kappa_4''.$$

Where going into the last line, we round 5.53 down to the value 5, which correspond to $\kappa_4'' \in \mathbf{K}''$.

4.3.5 Semantics of Function Application w.r.t Covenant-Judgement

The next rule defines how `train` changes Covenant-Judgement of the program output.

$$\texttt{train-L}\quad \frac{\Gamma \models \texttt{agnt}\ (a \to b) : \kappa_1, \Gamma \models \texttt{data}\ a : \kappa_2, \Gamma \models \texttt{data}\ b : \kappa_3, \kappa_1 \in \mathbf{K}, \kappa_2 \in \mathbf{K}', \kappa_3 \in \mathbf{K}''}{\Gamma \models \texttt{train}(\texttt{data}\ a, \texttt{data}\ b, \texttt{agnt}\ (a \to b)) :\ \exists \kappa_3 \in \mathbf{K}\ \text{s.t. } \kappa_3 = \texttt{proj}(\kappa_1 \otimes \kappa_2 \otimes \kappa_3)}$$

$$\text{where } \texttt{proj}(\kappa_1 \otimes \kappa_2 \otimes \kappa_3) := \text{arith}^{-1}\left(\frac{\text{arith}(\kappa_1) \cdot \text{arith}(\kappa_2) \cdot \text{arith}(\kappa_3)}{\text{arith}(\mathbf{K}) \cdot \text{arith}(\mathbf{K}') \cdot \text{arith}(\mathbf{K}'')}\right).$$

In `train-L` the `train` function `train`s some `agnt` $(a \to b)$ using `data` a and `data` b, each one judged as κ_1 and κ_2. In this case, the `train`ed `agnt` $(a \to b)$ is in the product space of the three Covenant-Judgements. The last rule defines how `infer` changes Covenant-Judgement of the program output.

$$\texttt{infer-L}\ \frac{\Gamma \models \texttt{agnt}\ (a \to b) : \kappa_1, \Gamma \models \texttt{data}\ a : \kappa_2, \kappa_1 \in \mathbf{K}, \kappa_2 \in \mathbf{K}'}{\Gamma \models \texttt{infer}(\texttt{data}\ a, \texttt{agnt}\ (a \to b)) :\ \exists \kappa_3 \in \mathbf{K}''\ \text{s.t. } \kappa_3 = \texttt{proj}(\kappa_1 \otimes \kappa_2)}$$

$$\text{where } \texttt{proj}(\kappa_1 \otimes \kappa_2) := \text{arith}^{-1}\left(\frac{\text{arith}(\kappa_1)\cdot\text{arith}(\kappa_2)}{\text{arith}(\mathbf{K})\cdot\text{arith}(\mathbf{K}')}\right).$$

In `infer`-L, the `infer` function runs some `agnt` $(a \to b)$ against input **`data`** a to output prediction or response **`data`** b. The resultant **`data`** b is in the product space.

▶ **Example 30.** (**`train`-L Composition**) Suppose we have this set of (ρro)Metheus computational `asset` s, and their Covenant-Judgement:

$$\Gamma \models \texttt{agnt}\ (a \to b) : \kappa \in \mathbf{K}, \quad \text{s.t. arith}(\kappa) = 5, \quad \text{and arith}(\mathbf{K}) = 10, \quad \mathbf{K} = \{\kappa_0, \ldots, \kappa_9\}.$$
$$\Gamma \models \texttt{data}\ a : \kappa' \in \mathbf{K}', \quad \text{s.t. arith}(\kappa') = 25, \quad \text{and arith}(\mathbf{K}') = 30.$$
$$\Gamma \models \texttt{data}\ b : \kappa'' \in \mathbf{K}'', \quad \text{s.t. arith}(\kappa'') = 18, \quad \text{and arith}(\mathbf{K}'') = 20.$$

Then we know that under `train`ing, the Covenant-Judgement of the output `agnt` $(a \to b)'$ is:

$$\texttt{proj}(\kappa \otimes \kappa' \otimes \kappa'') := \frac{5 \cdot 25 \cdot 18}{10 \cdot 30 \cdot 20} \cdot 10 = 3.75$$
$$\text{let } \texttt{train}(\texttt{data}\ a, \texttt{data}\ b, \texttt{agnt}\ (a \to b)) := \texttt{agnt}\ (a \to b)'$$
$$\Gamma \models \texttt{agnt}\ (a \to b)' : \kappa_2 \in \mathbf{K}'.$$

Where going into the last line, we round down to the value 3, which correspond to $\kappa_2 \in \mathbf{K}$. Observe that training on lower quality data lowers the quality of the model `agnt` $(a \to b)$.

▶ **Example 31.** (**`infer`-L Composition**) Suppose we have this set of (ρro)Metheus computational `asset` s, and their Covenant-Judgement:

$$\Gamma \models \texttt{agnt}\ (a \to b) : \kappa \in \mathbf{K}, \quad \text{s.t. arith}(\kappa) = 5, \quad \text{and arith}(\mathbf{K}) = 10,$$
$$\Gamma \models \texttt{data}\ a : \kappa' \in \mathbf{K}', \quad \text{s.t. arith}(\kappa') = 19, \quad \text{and arith}(\mathbf{K}') = 30,$$
$$\mathbf{K}'' = \{\kappa_0, \ldots, \kappa_{24}\}.$$

Then we know that under `infer`ence, the Covenant-Judgement of the output **`data`** b is:

$$\texttt{proj}(\kappa \otimes \kappa') := \frac{5 \cdot 19}{10 \cdot 30} \cdot 24 = 22.799$$
$$\text{let } \texttt{infer}(\texttt{data}\ a, \texttt{agnt}\ (a \to b)) := \texttt{data}\ b$$
$$\Gamma \models \texttt{data}\ b : \kappa_{21} \in \mathbf{K}''.$$

Where going into the last line, we round 22.799 down to the value 22, which correspond to $\kappa_{21} \in \mathbf{K}''$.

▶ Remark 32. (**Computational efficiency of `seqd`-L, `train`-L, and `infer`-L Composition**) Observe that in the examples above, we used simple arithmetic to find the Covenant-Judgement of the composed value. Notably, in example 30 we did not run the `train` function, which would be quite expensive. [11] And in example 31, we did not run the `infer` function, whose cost is also not trivial. This satisfies the *static check* aspect of the (ρro)Metheus environment feature, whereby code that could potentially lead to "bad" effects are not run at all. In this case the "bad" outcome may include:

- inference on poor input data, leading to low quality output;

[11] Indeed a single training run may cost 10s of millions of USD.

- training on poor quality data, leading to lower quality model `agnt`.

▶ Remark 33. (**The role of** FairCovenant **Foundation in defining inductive** Loare-Logic judgements) Observe that the `proj()` function we defined for `train` can never improve the Covenant-Judgement of the trained `agnt`, it can only degrade its value. This `proj()` may be too stringent in production. In reality the stringency of `proj()` is context dependent, and will be defined in conjunction with stakeholders as work streams within the FairCovenant Foundation.

4.3.6 Proof of Covenant-Compliance with Loare-Logic

This section places the elementary concepts of Loare-Logic introduced in the previous section in the context of an example, and walk through how a (ρro)Metheus compiler uses Loare-Logic to enforce Covenant-Judgement compliance.

▶ **Example 34.** (**Proof of Covenant-Compliance**) Given Covenant-Judgements sets:

$$\text{data Covenant-Judgement}: \quad \mathbf{K}_a = \{\kappa_1^a, \ldots, \kappa_{10}^a\}, \quad \mathbf{K}_b = \{\kappa_1^b, \ldots, \kappa_{10}^b\}, \quad \mathbf{K}_c = \{\kappa_1^c, \ldots, \kappa_{10}^c\}, \quad \mathbf{K}_d = \{\kappa_1^d, \ldots, \kappa_{10}\},$$

$$\text{function Covenant-Judgement}: \quad \mathbf{K}_{f_1} = \{\kappa_1^{f_1}, \kappa_2^{f_1}, \kappa_3^{f_1}, \kappa_4^{f_1}, \kappa_5^{f_1}\}, \quad \mathbf{K}_{f_2} = \{\kappa_1^{f_2}, \kappa_2^{f_2}, \kappa_3^{f_2}\}, \quad \mathbf{K}_{f_3} = \{\kappa_1^{f_3}, \kappa_2^{f_3}, \kappa_3^{f_3}\}.$$

And available computational `asset` with Covenant-Judgement:

$$\begin{aligned}
&\Gamma \models \mathsf{x}_1 :: \mathtt{data}\ a : \kappa_9^a, \quad \Gamma \models \mathsf{x}_2 :: \mathtt{data}\ a : \kappa_6^a, \quad \Gamma \models \mathsf{x}_3 :: \mathtt{data}\ a : \kappa_7^a, \\
&\Gamma \models \mathsf{y} :: \mathtt{data}\ b : \kappa_8^b \\
&\Gamma \models \mathbf{fn}_1 :: \mathtt{agnt}\ (\mathtt{data}\ a \to \mathtt{data}\ c) : \kappa_5^{f_1}, \\
&\Gamma \models \mathbf{fn}_2 :: \mathtt{agnt}\ (\mathtt{data}\ c \to \mathtt{data}\ d) : \kappa_2^{f_2}.
\end{aligned} \tag{13}$$

Additionally, we know that there is some:

- $\kappa_{\mathsf{j}}^{f_3} \in \mathbf{K}_{f_3}$ s.t. $\Gamma \models \mathtt{agnt}\ (\mathtt{data}\ a \to \mathtt{data}\ d) : \kappa_{\mathsf{j}}^{f_3}$.
- $\kappa_{\mathsf{i}}^{d} \in \mathbf{K}_{d}$ s.t. $\Gamma \models \mathtt{asset}\ : \kappa_{\mathsf{i}}^{d}$.

Now suppose the user builds a Typed ρ-Calculus expression as follows:

$$\mathsf{d}_* = \underbrace{\mathtt{infer}\ \mathsf{x}_1\ \underbrace{\underbrace{\left(\mathtt{train}\ \underbrace{(\mathsf{x}_2 \diamond \mathsf{x}_3}_{(2)},\ \mathsf{y})\ \underbrace{(\mathbf{fn}_1 \rightarrowtail \mathbf{fn}_2)}_{(1)}\right)}_{(3)}}_{(4)}. \tag{14}$$

Prompting the program to train some model $\mathbf{fn}_1 \rightarrowtail \mathbf{fn}_2$ on dataset $(\mathsf{x}_2 \diamond \mathsf{x}_3, \mathsf{y})$, before doing inference on input x_1. Now given a client-defined set of boundary condition on the quality of the output to expression (14) with:

$$\kappa_*^d \geq \kappa_5^d.$$

Stating the output of expression (14) must witness some covenant-value greater than or equal to κ_5^d. Then we can construct a proof that witnesses the Covenant-Compliance of expression (14) as follows:

1. Using the rule `seqd-L`, function composition under $\rightarrowtail$ reduces to:

$$\texttt{seqd-L}\ \frac{\Gamma \models \mathbf{fn}_1 : \kappa_5^{f_1}, \quad \Gamma \models \mathbf{fn}_2 : \kappa_2^{f_2}}{\Gamma \models (\mathbf{fn}_1 \ \rightarrowtail\ \mathbf{fn}_2) : \kappa_3^{f_3}}$$

   Where we determine the value of $\kappa_3^{f_3}$ with:

$$\texttt{proj}(\kappa_5^{f_1} \ \otimes\ \kappa_2^{f_2}) = 3 \times \sqrt{\left(\frac{5}{5}\right)^2 + \left(\frac{2}{3}\right)^2} = 3.60 \sim 3.$$

   So we have in the conclusion: $\Gamma \models (\mathbf{fn}_1 \ \rightarrowtail\ \mathbf{fn}_2) : \kappa_3^{f_3}$.

2. Using the rule `compb-L`, we have:

$$\texttt{compa-L}\ \frac{\Gamma \models \mathsf{x}_2 : \kappa_6^a, \quad \Gamma \models \mathsf{x}_3 : \kappa_7^a}{\Gamma \models (\mathsf{x}_2 \ \diamond\ \mathsf{x}_3) : \kappa_6^a}$$

3. Using the rule `train-L`, we have:

$$\texttt{train-L}\ \frac{\Gamma \models (\mathsf{x}_2 \ \diamond\ \mathsf{x}_3) : \kappa_6^a, \quad \Gamma \models \mathsf{y} : \kappa_8^b, \quad , \Gamma \models (\mathbf{fn}_1 \ \rightarrowtail\ \mathbf{fn}_2) : \kappa_3^{f_3}}{\Gamma \models \texttt{train}\ (\mathsf{x}_2 \ \diamond\ \mathsf{x}_3, \mathsf{y})(\mathbf{fn}_1 \ \rightarrowtail\ \mathbf{fn}_2) : \kappa_2^{f_3}}$$

   Where we determined the value of projected Covenant-Judgement with:

$$\texttt{proj}(\kappa_6^a \ \otimes\ \kappa_8^b \ \otimes\ \kappa_3^{f_3}) = 3 \times \frac{6 \times 8 \times 3.60}{10 \times 10 \times 3} = 1.72 \sim 2.$$

4. Finally using `infer-L` we have:

$$\texttt{infer-L}\ \frac{\Gamma \models \mathsf{x}_1 : \kappa_9^a, \quad \Gamma \models (\mathbf{fn}_1 \ \rightarrowtail\ \mathbf{fn}_2) : \kappa_2^{f_3}}{\Gamma \models \mathsf{d}_* : \kappa_6^d}$$

   Where we inductively define the output Covenant-Judgement with:

$$\texttt{proj}(\kappa_9^a \ \otimes\ \kappa_2^{f_3}) = 10 \times \frac{9 \times 2}{10 \times 3} = 6.$$

   And so we have:

$$\Gamma \models \mathsf{d}_* : \kappa_6^d,$$

   witnessing the expected output value of expression (14) to be greater than the boundary condition of κ_5^d. Thus the (ρro)Metheus has used Loare-Logic to generate a proof stating the computation is Covenant-Compliant

We can rewrite this entire proof in shorter format with:

$$\frac{\dfrac{\dfrac{\Gamma \models \mathbf{fn}_1 : \kappa_5^{f_1} \qquad \Gamma \models \mathbf{fn}_2 : \kappa_2^{f_2}}{\Gamma \models (\mathbf{fn}_1 \ \rightarrowtail\ \mathbf{fn}_2) : \kappa_3^{f_3}} \qquad \dfrac{\Gamma \models \mathsf{x}_2 : \kappa_6^a \qquad \Gamma \models \mathsf{x}_3 : \kappa_7^a}{\Gamma \models (\mathsf{x}_2 \ \diamond\ \mathsf{x}_3) : \kappa_6^a} \qquad \Gamma \models \mathsf{y} : \kappa_8^b}{\Gamma \models \texttt{train}(\mathsf{x}_2 \ \diamond\ \mathsf{x}_3, \mathsf{y})\ (\mathbf{fn}_1 \ \rightarrowtail\ \mathbf{fn}_2) : \kappa_2^{f_2}} \qquad \Gamma \models \mathsf{x}_1 : \kappa_9^a}{\Gamma \models \mathsf{d}_* = (\texttt{infer}\ \mathsf{x}_1(\mathbf{fn}_1 \ \rightarrowtail\ \mathbf{fn}_2)) : \kappa_6^d}$$

This is known as a proof tree, or in this context: a Loare-Logic proof tree.

This exact proof is generated by the (ρro)Metheus compiler before any training or inference is conducted. So that if the proposed computation in expression (14) is Covenant-Compliant, then this proof can be produced as a witness. Otherwise the compiler halts the computation and asks the analyst to input higher quality `asset` into the Typed ρ-Calculus expression, so as to satisfy client-defined notions of quality. This is called static guarantee of Covenant-Compliance. The Typed ρ-Calculus language thus has satisfies the following desiderata:

- (*Monetary*) save money on training and inference by checking the *expected* quality of the final computation, *before* any expensive computation is done client side.
- (*Legality*): Loare-Logic has shifted the burden of proof from the analyst to the compiler. Thereby opening up a greater set of inference tasks that can be done by analysts in high-stake settings with legal responsibilities.
- (*Composability*): the proof tree generated by Loare-Logic used elementary definitions of Covenant-Judgement generated by The FairCovenant Foundation, and boundary conditions of inference quality defined by the client. It then inferred the final Covenant-Judgement of the computation without further human input, thereby greatly augmenting the capacity of regulators to audit the computation trees written by analysts deep in the bowels of their work routine.
- (*End-to-end-oversight*): although the computation in expression (14) appear simple and "one lined," it in fact may span many days or even weeks. As the `train` function alone may be an industrial training run of some deep learning model. The (ρro)Metheus compiler could audit this industrial routine a-priori in its entirety, or audit aspects of the computation over the course of its realization. This gives flexibility as there may be some discrepancy between the *expected* quality of the output under Covenant-Judgement, and its actual quality after a training run. This flexibility is particularly salient in the case of deep learning models, which feature nonconvex objective functions with indeterminate quality a-priori training.
- (*Flexibility*): in this example we used author-defined notions of `proj()` and arithmetization functions. However in real life, it would be the stakeholders that define said functions. They may dial up or down the restrictiveness of the proof at each step along the proof tree generated above. Thereby controlling the amount of computation they deem acceptable.

▶ **Example 35.** (Typed ρ-Calculus **program decorated by** Loare-Logic **triples**) We can rewrite example (34) by decorating the expression with Loare-Logic triples with:

$$
\begin{aligned}
&\left\{\left\{\Gamma \models \mathsf{x}_1 : \kappa_9^a, \mathsf{x}_2 : \kappa_6^a, \mathsf{x}_3 : \kappa_7^a, \mathsf{y} : \kappa_8^b, \mathbf{fn}_1 : \kappa_5^{f_1}, \mathbf{fn}_2 : \kappa_2^{f_2}\right\}\right\} \\
\mathsf{d}_* = \ &\texttt{infer} \\
&\quad \mathsf{x}_1 \\
&\quad \texttt{train}\ (\mathsf{x}_2 \diamond \mathsf{x}_3, \mathsf{y})\ \ (\mathbf{fn}_1 \rightarrowtail \mathbf{fn}_2) \\
&\quad \left\{\left\{\Gamma \models \kappa_*^d \geq \kappa_5^d\right\}\right\}
\end{aligned}
\tag{15}
$$

This conforms to the syntax presented in definition (26).

References

1 *Covenant*. Cornell Law, 2025.
2 *Covenant that runs with the land*. Cornell Law, 2025.
3 A. Zaldivar P. Barnes L. Vasserman B. Hutchinson E. Spitzer I. D. Raji T. Gebru M. Mitchell, S. Wu. Model cards for model reporting. *FAT*, (220-229), 2019. doi:10.1145/3287560.3287596.
4 Oddur Kjartansson Mahima Pushkarna, Andrew Zaldivar. Data cards: Purposeful and transparent dataset documentation for responsible ai. *FAccT*, 2022. doi:10.1145/3531146.3533231.
5 Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, 2002.
6 Benjamin C. Pierce. *Software Foundations, Volume 2: Programming Language Foundations*. The MIT Press, 2017.
7 Richard F. Schmidt. *Software Engineering Architecture-driven Software Development*. Elsevier Inc, 2013.
8 Ling Xiao. *Metheus: Fair Accountable Transparent Machine Intelligence Audit*. 2025.
9 Ling Xiao and Warwick Powell. *FairCovenant: the Operating System of Trade, Talent, & Technology for the Global South*. 2025.