TITLE OF THE INVENTION

Agentic Compliance and Policy Governor For AI and Code With Machine Readable Proof Carrying Artefacts

ABSTRACT

A computer implemented compliance governance system is disclosed. Textual regulations and organisational policies are compiled into a dual representation of executable checks and normative rules including strict rules, defeasible rules and defeaters with explicit priorities. One or more generator agents produce candidate artefacts such as source code or AI generated content together with provenance metadata. One or more prosecutor agents execute the executable checks and synthesise counterexamples to construct violation arguments. An adjudicator constructs an argumentation graph over compliance and violation arguments induced by the normative rules and computes an extension under grounded acceptability semantics of an abstract argumentation framework in order to determine which arguments are accepted. When no violation argument remains accepted, a machine readable proof bundle is assembled that enumerates satisfied rules, defeated challenges and evidence traces, and the proof bundle is cryptographically signed and attached to the artefact, thereby yielding a proof carrying artefact. When at least one violation argument remains accepted, generator guidance is computed based on least margin or highest risk rules so that the generator agents revise the candidate artefact and the process iterates. The invention provides automated, auditable and machine verifiable compliance for AI systems and software in regulated environments and integrates with development and deployment pipelines.

FIELD OF THE INVENTION

The invention relates to automated compliance and policy governance for software systems and artificial intelligence generated artefacts. More particularly it concerns systems and methods for enforcing and proving adherence to policies, regulations and best practices using multi agent orchestration, policy as code, structured argumentation and proof carrying artefacts.

BACKGROUND OF THE INVENTION

Regulated industries such as financial services, healthcare and government increasingly rely on software systems and artificial intelligence models to perform critical functions. Regulators expect those systems to comply with laws, sector regulations, security standards and internal policies and expect organisations to provide evidence of such compliance on demand.

Traditional compliance workflows for software and AI are largely manual. They rely on static documents, checklists, manual code reviews and retrospective audits. These practices are slow, expensive and error prone. They do not scale to modern continuous integration and continuous delivery pipelines and they struggle to keep pace with evolving regulations and the rapid adoption of generative AI which can autonomously produce large volumes of code and content.

Policy as code frameworks have emerged, for example engines that evaluate declarative rules over configuration data and input requests. These engines are typically integrated into admission control points, build pipelines or API gateways. They normally evaluate policies in a single step for permit or deny decisions and may write logs of such decisions. They do not, however, provide a general mechanism to drive artefact generation towards compliance in an iterative loop nor do they emit a portable, machine verifiable proof that a particular artefact satisfies all relevant rules.

Formal methods and proof carrying code techniques have shown that it is possible to attach proofs of safety to low level programs so that a host can mechanically verify that a safety policy is satisfied before loading the code. Those techniques have generally been applied to low level properties such as memory safety and have not been adopted widely in commercial development. They also assume that someone or some specialised tool constructs the proof as part of compilation. They do not integrate with policy as code, multi agent architectures or AI generators.

There is therefore a need for a system that can operate over higher level policies, that can automatically drive AI and software artefact generation towards compliance, that can resolve policy conflicts in a principled way and that can produce for each artefact a machine readable, cryptographically verifiable proof of compliance suitable for audits and automated verification.

SUMMARY OF THE INVENTION

According to a first aspect there is provided a computer implemented compliance governance system comprising one or more processors and memory storing instructions that implement:

- A policy compiler which receives textual regulations and organisational policies and compiles them into a dual representation comprising executable checks and a set of normative rules including strict rules, defeasible rules and defeaters with explicit priorities, the normative rules being stored in a policy knowledge base.
- One or more generator agents which receive user specifications and policy identifiers and in response generate candidate artefacts comprising source code or natural language content together with provenance metadata.
- One or more prosecutor agents which receive the candidate artefacts, execute the executable checks and synthesise counterexamples to construct violation arguments alleging breaches of the normative rules.

- An adjudicator which constructs an argumentation graph over compliance arguments and violation arguments induced by the normative rules and the prosecutor outputs, and which computes an extension under grounded acceptability semantics of an abstract argumentation framework in order to determine a set of accepted arguments.
- A proof bundle builder which, when the set of accepted arguments contains no violation argument, assembles a machine readable proof bundle that records identifiers and statuses of rules, accepted and rejected arguments, evidence traces and at least one cryptographic signature, and attaches the proof bundle to the artefact so as to create a proof carrying artefact.

- A generator guidance component which, when the set of accepted arguments contains at least one violation argument, computes guidance identifying rules with least margin or highest risk and causes the one or more generator agents to revise the candidate artefact so that the process iterates.

According to a second aspect there is provided a method performed by one or more processors for generating a proof carrying artefact. The method comprises compiling textual regulations into executable checks and normative rules, generating a candidate artefact and provenance metadata by at least one generator agent, executing the executable checks and synthesising counterexamples by at least one prosecutor agent to construct violation arguments, constructing an argumentation graph by an adjudicator and computing a grounded extension to determine accepted arguments, and when no violation argument is accepted assembling and signing a machine readable proof bundle and binding it to the artefact, and when at least one violation argument is accepted computing generator guidance that targets rules with least margin or highest risk and causing the artefact to be revised and the method to repeat until no violation argument is accepted.

According to further aspects there are provided a non transitory computer readable medium storing instructions which implement the method, a proof carrying artefact comprising payload content and an attached proof bundle as described, and a network service that exposes an interface to receive artefacts and policies and returns proof bundles or generator guidance.

The invention thus transforms compliance checking from a one shot gate into a multi agent optimisation loop that yields artefacts that are compliant by construction together with machine verifiable proof of such compliance.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a high level architectural block diagram of an agentic compliance and policy governor system.

Figure 2 is a sequence diagram illustrating an example interaction between a client, generator agent, prosecutor agent, adjudicator, proof bundle builder and deployment gate.

Figure 3 is a flow diagram of a policy compilation pipeline.

Figure 4 is a flow diagram of a generation, prosecution, adjudication and guidance loop.

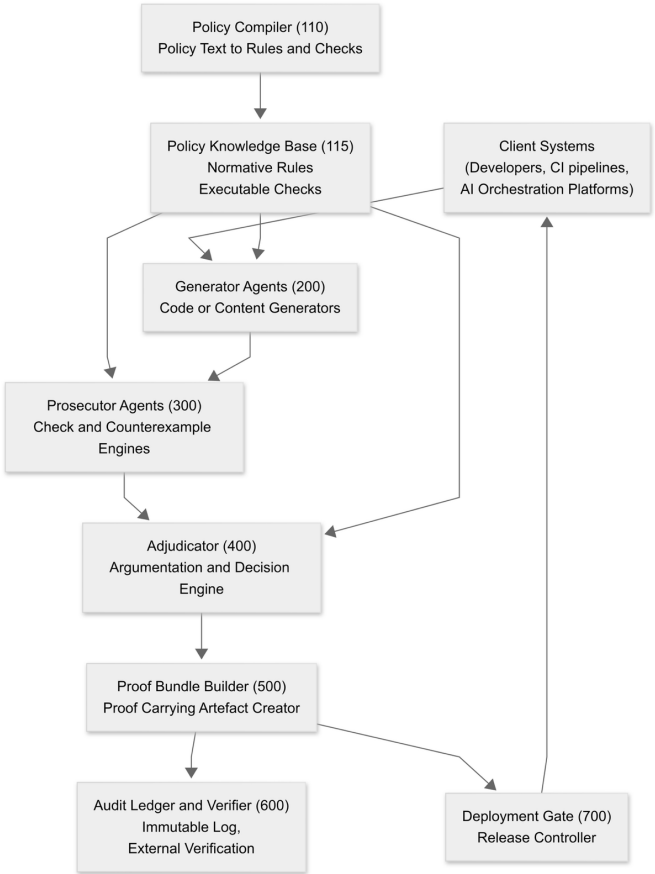
Figure 5 is a conceptual diagram of an argumentation graph used by the adjudicator.

Figure 6 is a diagram illustrating construction of a proof carrying artefact and use of a proof bundle at a deployment gate and external verifier.

DETAILED DESCRIPTION OF EMBODIMENTS

Overall Architecture

Figure 1 shows an embodiment of an agentic compliance and policy governor system.



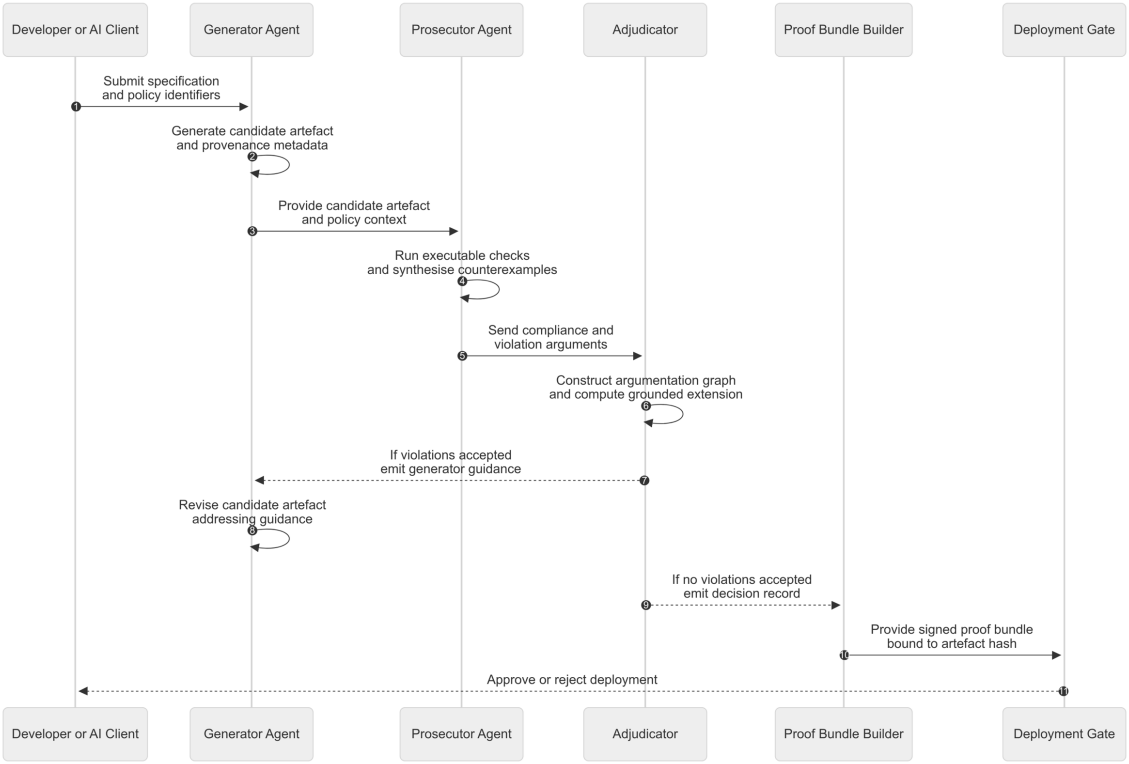
Client systems include developer tools, continuous integration servers, model orchestration platforms and other systems that wish to obtain compliant artefacts. These clients provide specifications of desired functionality together with identifiers of applicable policy sets.

The policy compiler parses textual policy artefacts such as regulations, standards and internal guidelines and emits executable checks and normative rules which are stored in the policy knowledge base. The knowledge base is referenced by generator agents, prosecutor agents and the adjudicator.

Generator agents produce candidate artefacts according to the client specification and with awareness of the policy context. Prosecutor agents apply executable checks and generate counterexamples to discover possible violations. The adjudicator consumes arguments from the knowledge base and prosecutor findings and computes a compliance decision. The proof bundle builder composes a proof and binds it to the artefact. The audit ledger stores proof bundles and artefact hashes for later verification. The deployment gate enforces that only artefacts with valid and unexpired proof bundles are released.

Interaction Sequence

Figure 2 illustrates an example interaction sequence from the point of view of a developer or AI client.



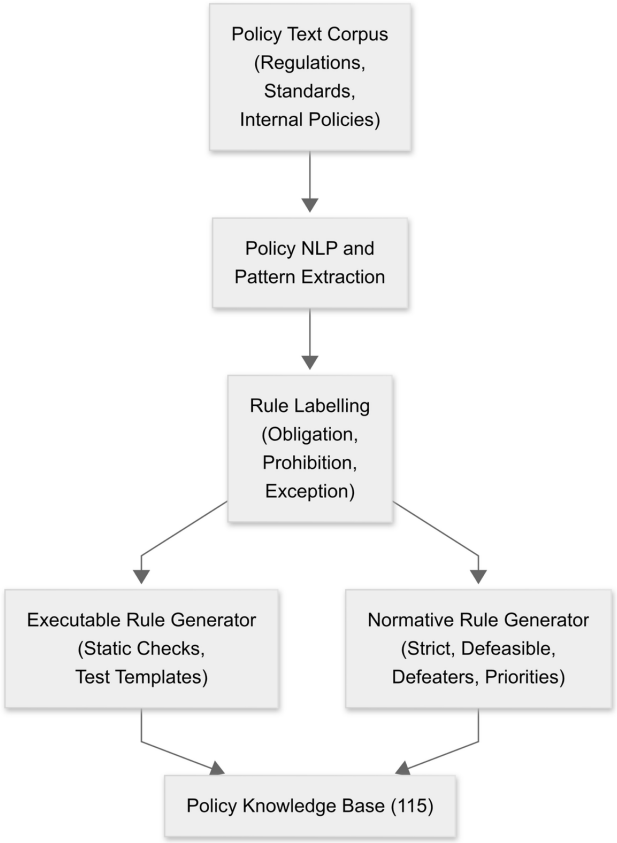
In step one the developer or AI orchestration submits the desired functionality specification and references to policy sets. The generator agent constructs a first candidate artefact, for example source code modules, infrastructure as code templates or natural language content. The generator attaches provenance metadata such as model prompts, base models, training data identifiers or repository commit identifiers.

The candidate artefact is passed to one or more prosecutor agents. These execute policy specific checks including static analysis, configuration evaluation, dynamic tests, fuzzing, property based test generation and content scanning. From the results of these checks and from the normative rules, the prosecutors construct arguments that assert either satisfaction of rules or violations, together with counterexamples or evidence.

The adjudicator then builds an argumentation graph and computes a grounded extension to determine which arguments are accepted. If any accepted argument alleges a violation of a rule of sufficient priority, generator guidance is emitted and the process returns to the generator which revises the artefact. When no violation argument remains accepted the adjudicator passes its decision record to the proof bundle builder, which constructs a signed proof bundle. The deployment gate verifies the proof bundle signature and integrity and either approves or blocks deployment accordingly.

Policy Compiler and Knowledge Base

Figure 3 illustrates an embodiment of a policy compilation pipeline.

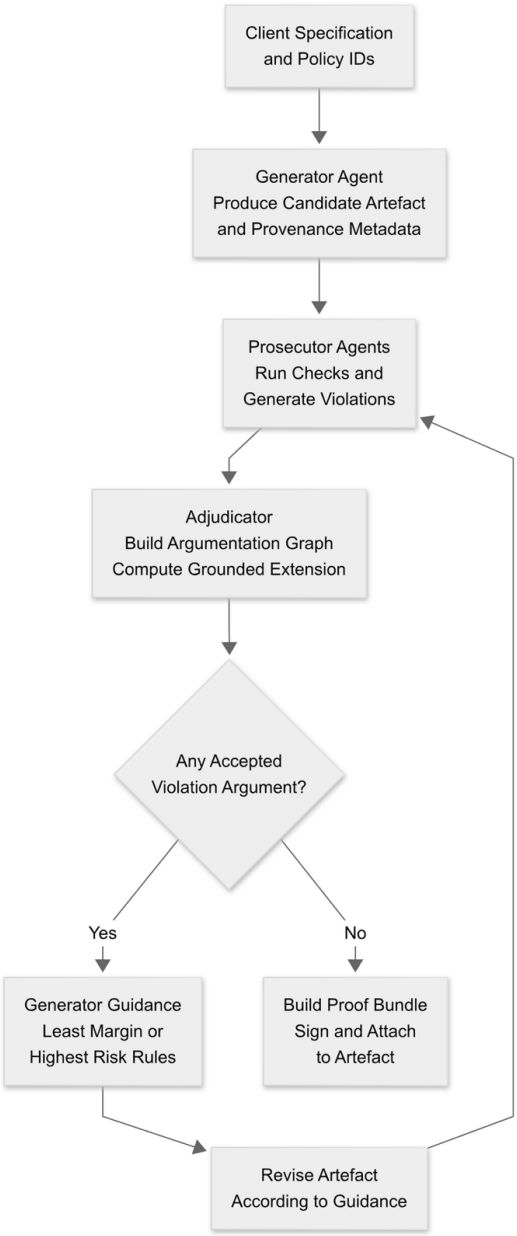


Textual policy material is processed by a natural language processing module which identifies candidate clauses expressing obligations, prohibitions, conditions and exceptions. A rule labelling stage classifies each clause by modality and strength, for example strict obligation, soft recommendation, exception condition or pure defeater.

From the labelled clauses the executable rule generator produces concrete check definitions such as static code analysis rules, configuration queries and parameterised test templates. In parallel the normative rule generator emits formal logic rules that capture the normative structure using a representation that distinguishes strict rules, defeasible rules and defeaters and that attaches explicit priorities. Both executable rules and normative rules are stored in the policy knowledge base keyed by policy identifiers and rule identifiers. The knowledge base thereby supports both test execution and argumentation.

Generation, Prosecution and Adjudication Loop

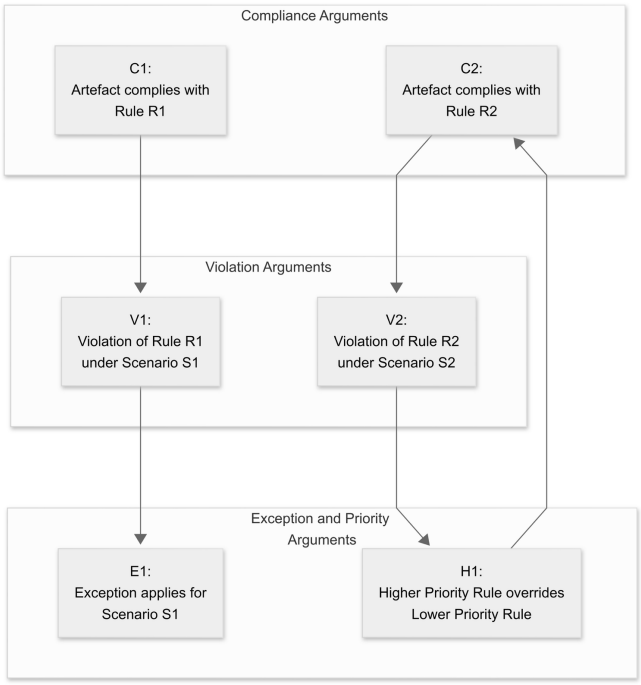
Figure 4 presents a logical flow of the compliance loop.



Given a client specification and policy identifiers, the generator agent produces an initial candidate artefact and metadata. Prosecutor agents execute relevant checks and create violation arguments for any failed checks or unresolved rules. The adjudicator builds an argumentation graph and computes a grounded extension. A decision node determines whether any accepted argument represents a rule violation. If so, generator guidance is computed. The guidance includes rule identifiers and locations within the artefact associated with least compliance margin or highest risk. The generator uses the guidance to produce a revised artefact, after which checks and adjudication repeat. If no violation argument remains accepted, a proof bundle is constructed and signed and is attached to the artefact.

Argumentation Graph Structure

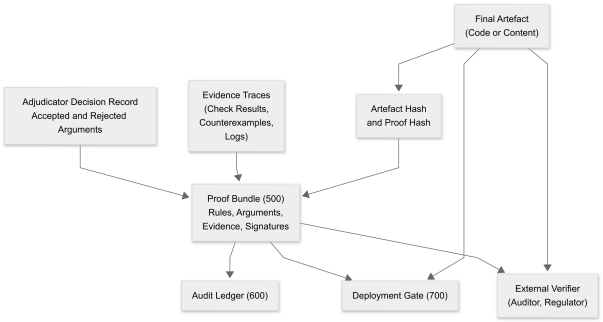
Figure 5 depicts an abstract view of the argumentation graph used by the adjudicator.



Each node represents an argument derived from premises and rules. Compliance arguments C1, C2 assert that the artefact satisfies particular rules. Violation arguments V1, V2 assert that in specific scenarios a rule is breached. Exception argument E1 undercuts violation V1 by asserting that an exception condition holds. Priority argument H1 undercuts violation V2 and supports compliance argument C2 by asserting that a higher priority rule renders the apparent violation non decisive. Directed edges indicate attacks. The adjudicator computes the grounded extension of this graph to decide which of C1, C2, V1, V2, E1, H1 are accepted. Accepted arguments determine the compliance status of each rule.

Proof Carrying Artefact and Deployment

Figure 6 illustrates construction and use of a proof carrying artefact.



The final artefact is accompanied by an adjudicator decision record and evidence traces from prosecutor agents. The proof bundle builder packages these along with the policy rule identifiers and one or more digital signatures into a proof bundle. A cryptographic hash binds the proof bundle to the exact artefact version. The bundle is stored in an audit ledger for long term traceability and is provided to the deployment gate which verifies the signature and checks that no required rule is marked violated. The proof bundle and artefact can later be

supplied to an external verifier which re evaluates the argumentation or checks the internal consistency of the bundle.

Implementation Details

In one embodiment generator agents are implemented as wrappers around large language models configured to generate code, configuration or natural language content. The generator wrapper receives not only a functional specification but also policy identifiers and guidance hints. The wrapper constructs prompts that describe the relevant rules in natural language and that include previous violation messages and counterexamples so that the model is steered away from prior issues. The generator records provenance metadata including base model identifiers, system prompts and user prompts.

Prosecutor agents may be implemented as separate microservices responsible for particular policy domains. A security prosecutor executes static analysis and dynamic tests for secure coding rules. A privacy prosecutor scans for personal data fields and checks that handling is compliant with privacy policies. A fairness prosecutor evaluates model outputs for bias metrics. Each prosecutor exposes an interface that accepts artefacts and returns structured results including pass or fail statuses, locations in the artefact, and any generated test inputs that demonstrate failures.

The adjudicator uses a structured argumentation calculus. Arguments are constructed from facts about the artefact, for example outputs from prosecutors and metadata, combined with normative rules from the policy knowledge base. Rules are categorised into strict rules and defeasible rules. Defeater rules only attack other rules. Priority relations between rules are encoded so that attacks from higher priority rules can undercut lower priority ones. The argumentation graph is represented as a directed graph with nodes representing arguments and edges representing attacks. The grounded extension is computed using a fixpoint algorithm. The algorithm begins with all arguments that have no attackers, marks them as tentatively accepted, and iteratively eliminates attacked arguments and accepts arguments whose attackers have all been eliminated until convergence.

Generator guidance is computed by examining violation arguments that remain accepted. For each such rule the system computes a margin measure. For numeric thresholds this may be a distance from the threshold. For non numeric rules the system may consider the number of attempts the prosecutor required to find a counterexample, the coverage of tests, or an assigned risk weight. A selection policy chooses one or more rules to address in the next iteration. This selection may use a multi armed bandit strategy where each rule is an arm with an estimated reward equal to expected compliance improvement weighted by risk. Alternatively a multi objective optimisation approach may be used where the fitness function combines functional utility of the artefact and compliance level. The guidance sent to the generator includes the selected rule identifiers, locations in the artefact relevant to each rule and descriptions of counterexamples.

Proof bundles are encoded in a structured format such as JSON or XML. Each bundle contains a header with policy set identifiers and artefact identifiers, a section listing each rule with status, a section detailing arguments and attacks that participated in the grounded extension, and a section containing evidence references and hashes. Bundles are signed using a private key held in secure hardware such as a hardware security module. Verification consists of checking signatures, confirming that all mandatory rules are marked satisfied or

overridden by accepted higher priority rules, and optionally recomputing the grounded extension from the record.

The system can be deployed in on premises, cloud or hybrid environments. In one implementation each agent type is packaged as a container and orchestrated by a Kubernetes cluster. A message queue or event bus routes artefacts and arguments between agents. The policy knowledge base is stored in a database with versioning so that decisions can be reproduced for earlier policy versions.

CLAIMS

1. A computer implemented compliance governance system comprising one or more processors and a memory storing instructions which, when executed by the one or more processors, cause the system to perform operations comprising:

compiling, by a policy compiler, textual regulations and organisational policies into a dual representation comprising executable checks and a set of normative rules including strict rules, defeasible rules and defeaters, the normative rules being associated with explicit priorities and stored in a policy knowledge base;

generating, by at least one generator agent, a candidate artefact and provenance metadata in response to a client specification and one or more policy identifiers, the candidate artefact comprising source code or natural language content;

prosecuting, by at least one prosecutor agent, the candidate artefact by executing the executable checks referenced from the policy knowledge base and synthesising counterexamples, thereby constructing violation arguments alleging violations of the normative rules;

adjudicating, by an adjudicator, the violation arguments and compliance arguments derived from the normative rules by constructing an argumentation graph whose nodes represent arguments and whose directed edges represent attacks between arguments, and computing an extension of the argumentation graph under grounded acceptability semantics of an abstract argumentation framework to determine a set of accepted arguments;

in response to a determination that the set of accepted arguments contains no violation argument, assembling, by a proof bundle builder, a machine readable proof bundle that enumerates rules checked and their statuses, accepted and rejected arguments, evidence traces and at least one cryptographic signature, and associating the proof bundle with the candidate artefact so as to yield a proof carrying artefact; and

in response to a determination that the set of accepted arguments contains at least one violation argument, computing generator guidance that identifies rules having least compliance margin or highest compliance risk and causing the at least one generator agent to revise the candidate artefact in accordance with the generator guidance, wherein the prosecuting and adjudicating are repeated on the revised candidate artefact until the set of accepted arguments contains no violation argument.

2. The system of claim 1, wherein the policy compiler applies natural language parsing and semantic analysis to extract obligations, prohibitions, conditions and exceptions from the textual regulations and policies, labels each extracted clause as a strict rule, a defeasible rule or a defeater, and generates from each clause both an executable check definition and a corresponding normative rule entry in the policy knowledge base.
3. The system of claim 1, wherein the priorities encode a hierarchy between external regulations, industry standards and internal guidelines, and the adjudicator enforces the hierarchy by treating attacks arising from higher priority rules as defeating attacks against arguments arising from lower priority rules.
4. The system of claim 1, wherein the at least one prosecutor agent employs one or more of static program analysis, configuration evaluation, fuzz testing, property based test

generation and adversarial input synthesis to obtain counterexamples and to generate violation arguments.

5. The system of claim 1, wherein the adjudicator computes the grounded extension of the argumentation graph by iteratively accepting arguments without attackers, rejecting arguments attacked by accepted arguments and accepting further arguments whose attackers are all rejected until a fixpoint is reached.
6. The system of claim 1, wherein the proof bundle comprises rule identifiers, rule statuses, representations of accepted arguments, representations of rejected arguments, references to counterexample inputs and execution traces, cryptographic hash digests of artefact versions and one or more digital signatures over the proof bundle and artefact hash.
7. The system of claim 1, wherein the at least one generator agent comprises a large language model configured to generate source code or natural language content conditioned on policy information, and wherein the generator guidance comprises natural language instructions and counterexample descriptions supplied to the large language model.
8. The system of claim 1, wherein the generator guidance is produced by a selection policy that treats unresolved rules as arms in a multi armed bandit process and selects a next rule or artefact region to modify based on an expected improvement in compliance weighted by a risk weight assigned to each rule.
9. The system of claim 1, wherein revising the candidate artefact is guided by a multi objective optimisation that trades off task utility and compliance risk using a fitness function combining functional quality metrics and the number or severity of remaining violations.
10. The system of claim 1, wherein multiple heterogeneous prosecutor agents operate in parallel over disjoint subsets of rules or artefact regions and the adjudicator aggregates arguments from all prosecutor agents into the argumentation graph.
11. The system of claim 1, further comprising a deployment gate that receives the proof carrying artefact, verifies the cryptographic signature over the proof bundle and artefact hash, and prevents deployment of the artefact if a valid proof bundle is absent, the signature is invalid, or any non overridable rule is marked violated.
12. The system of claim 1, wherein an update to the textual regulations or organisational policies causes the policy compiler to regenerate only those normative rules and executable checks affected by the update and wherein the adjudicator reuses unaffected arguments when recomputing compliance for artefacts under the updated rules.
13. The system of claim 1, wherein the dual representation comprises a declarative policy language for the executable checks and a structured argumentation calculus for the normative rules and wherein rule identifiers are shared between the two representations.
14. The system of claim 1, wherein the proof bundle is verifiable by an external verifier without access to the prosecutor agents by re evaluating the argumentation graph over the rules, arguments and evidence encoded in the proof bundle.
15. The system of claim 1, wherein the adjudicator emits a machine readable decision record that lists for each rule the accepted arguments, rejected arguments and rule priorities used in deriving the decision.
16. A computer implemented method for generating a proof carrying artefact, the method comprising:

compiling textual regulations and organisational policies into executable checks and a set of normative rules including strict rules, defeasible rules and defeaters with explicit priorities, and storing the checks and rules in a policy knowledge base;

producing, by at least one generator agent, a candidate artefact and provenance metadata in response to a client specification and policy identifiers;

executing, by at least one prosecutor agent, the executable checks against the candidate artefact and synthesising counterexamples that instantiate violation arguments alleging breaches of the normative rules;

constructing, by an adjudicator, an argumentation graph over violation arguments and compliance arguments induced by the normative rules and facts about the candidate artefact, and computing a grounded extension of the argumentation graph to determine which arguments are accepted;

when no violation argument is accepted in the grounded extension, assembling a machine readable proof bundle that records rule identifiers and statuses, arguments and evidence traces, cryptographically signing the proof bundle and binding the proof bundle to the candidate artefact so as to create a proof carrying artefact; and

when at least one violation argument is accepted in the grounded extension, computing generator guidance that targets rules having least compliance margin or highest compliance risk, revising the candidate artefact in accordance with the generator guidance, and repeating the executing, constructing and computing until no violation argument is accepted.

17. The method of claim 16, wherein constructing the argumentation graph comprises associating each attack between arguments with a rule priority and resolving mutual attacks by treating attacks from arguments derived from higher priority rules as defeating attacks against arguments derived from lower priority rules.
18. The method of claim 16, wherein the least compliance margin for a numeric threshold rule is computed as a distance between an observed metric and the threshold and wherein the least compliance margin for a non numeric rule is estimated from a measure of search effort required for the prosecutor agent to find a counterexample.
19. The method of claim 16, further comprising writing the signed proof bundle and a hash of the artefact to an append only audit log for later verification.
20. The method of claim 16, further comprising embedding a verifier token that references the proof bundle in a software artefact manifest or a document header of the artefact.
21. The method of claim 16, wherein the at least one generator agent is an artificial intelligence model trained to condition generation on policy features or rule identifiers and wherein the generator guidance comprises identifiers of violated rules, descriptions of counterexamples and natural language hints.
22. The method of claim 16, wherein executing the checks and synthesising counterexamples is performed for both source code artefacts and natural language content artefacts using policy specific test harnesses.
23. A non transitory computer readable medium storing instructions that, when executed by one or more processors, cause performance of the method of any one of claims 16 to 22.

24. The non transitory computer readable medium of claim 23, wherein the instructions further cause computing of the grounded extension of the argumentation graph as a least fixpoint and signing of the proof bundle with a key held in a hardware security module.
25. A proof carrying artefact comprising:

a payload comprising source code, configuration data or natural language content generated under a policy context; and

an attached proof bundle that includes policy identifiers, rule representations, argumentation outcomes, evidence traces and a digital signature,

wherein the proof bundle attests that, when the payload is evaluated against the included rules and evidence using grounded acceptability semantics of an abstract argumentation framework, no violation argument remains accepted for non overridable rules.

26. The artefact of claim 25, wherein the proof bundle further includes an embedded verifier module that reconstructs the argumentation graph from the bundle and recomputes the grounded extension for verification.
27. The artefact of claim 25, wherein the digital signature binds the proof bundle to a hash of the payload and to a timestamp representing a compliance decision time.
28. The artefact of claim 25, wherein the proof bundle includes records of prior violations and their resolutions across iterations together with links to corresponding evidence traces.
29. A network service comprising one or more interfaces to receive candidate artefacts and policy specifications and one or more processors configured to execute instructions which implement the system of any one of claims 1 to 15 and to return to a client either a proof bundle associated with a proof carrying artefact or generator guidance for revising a non compliant artefact.
30. The system of claim 1, wherein the operations reduce a false negative rate for compliance violations and a time to compliance relative to executing the executable checks without adjudication under the abstract argumentation framework.

Patentability assertions, novelty and non obviousness

A. The dual compilation of textual policy into two coordinated artefacts, namely executable checks and a structured set of normative rules that include strict rules, defeasible rules, and defeaters with explicit priorities, is not taught by common policy as code engines that evaluate only executable predicates. The integration permits logical adjudication of exceptions and conflicts, not merely permit or deny outcomes.
B. The specific agentic loop that couples a generator with one or more prosecutor agents and an adjudicator that resolves attacks under grounded acceptability semantics, then emits a signed, machine readable proof bundle, and iterates until no violation argument remains accepted, is a distinctive workflow. Prior tools gate builds or flag violations, they do not produce a proof carrying artefact with formal acceptability semantics and a cryptographic decision record.
C. The feedback mechanism that targets rules with least margin or highest risk, including quantitative margin from thresholds and search effort, and that optionally uses multi armed bandit selection or multi objective optimisation to guide the next generation step, provides a technical improvement that reduces iterations and missed violations. This targeted guidance

for content or code generation under policy constraints is not present in standard continuous integration gatekeeping.
D. The proof bundle structure, which encodes the argumentation graph outcome, satisfied and defeated arguments, counter example traces, provenance, and signatures sufficient for independent machine verification without re running the prosecutors, exceeds conventional audit logs. Conventional tools record policy evaluations, they do not ship a self contained, verifiable proof aligned to abstract argumentation semantics.
E. The combination produces a concrete improvement to computer functionality. It transforms compliance from a brittle, manual checklist into an automated reasoning pipeline that yields deterministic, machine verifiable judgements with lower false negative rates and lower time to compliance. This is not an abstract idea performed mentally, it is a specific architecture that materially changes how software and content are generated and validated on computers.