AALBORG UNIVERSITY
STUDENT REPORT

Department of Electronic Systems
Aalborg University
http://www.aau.dk

Thesis Title:

Semantic Policy Interaction Analysis for Microsoft Entra ID Conditional Access Baselines

Thesis Theme:

Cyber Security

Thesis Period:

February 2026 to June 2026

Participant(s):

Sayed Mohammad Jalali

Supervisor(s):

Edlira Dushku

Page Numbers: 75

Date of Completion:

June 3, 2026

Abstract:

Cloud identity platforms increasingly act as the control plane for organizational security. In Microsoft Entra ID, effective access enforcement depends not only on individual Conditional Access policies, but also on how baseline policies, surrounding policies, and exclusions interact over time. This thesis examines semantic baseline evaluation, where an unchanged baseline policy artifact may still lose its intended governance effect because of changes in the surrounding policy structure.

The thesis proposes a bounded semantic evaluation approach for Microsoft Entra ID Conditional Access baselines. It separates direct baseline artifact comparison from semantic evaluation of surrounding policy composition, introduces a model for representing baseline intent and policy state, and implements the approach in an executable prototype. The prototype is evaluated using controlled scenario packages. The results show that it preserves the selected outcome distinctions and produces the expected records across the scenario set. The contribution is a structured and inspectable method for deriving semantic policy interaction candidates and structural coverage observations within a defined abstraction boundary.

The content of this report is freely available, but publication (with reference) may only be pursued in agreement with the authors through their consent.

Contents

Glossary of Terms and Abbreviations **4**

1 Introduction **8**

1.1 Motivation and Problem Statement 8

1.2 Scope and Research Questions 9

1.3 Thesis Contributions 9

2 Background **11**

2.1 Identity as a Control Plane in Cloud Architectures 11

2.2 Identity Governance in Microsoft Entra ID 11

2.3 Declarative Configuration as Security State 13

2.4 Compositional Interaction and Drift in Identity Systems 13

3 Literature Review **15**

3.1 State-Based Evaluation and Drift Detection 15

3.2 Compositional Exposure and Attack Path Reasoning 16

3.3 Declarative Security and Compliance as Code 16

3.4 Identity Conformance: Protocol Correctness to Tenant Governance 17

3.5 Identity Baseline Governance Gap 18

3.6 Positioning of This Thesis 19

4 Methodology **20**

4.1 Research Paradigm and Rationale 20

4.2 Research Design and Alignment 21

4.3 Scope, Assumptions, and Boundaries 21

4.4 Evaluation Strategy 22

4.5 Methodological Limitations 22

5 Architecture of the Proposed Framework **23**

5.1 Semantic Interaction Taxonomy 25

5.2 Abstraction Model for Semantic Evaluation 27

5.2.1 Intent Representation 27

5.2.2 Analytical Boundary and Dimensions 29

5.2.3 Policy Representation and Governed Scope 30

5.2.4 Mechanism Specific Modeling Assumptions 32

5.3 Semantic Screening Procedure 33
5.4 Derived Entry Schema 39
5.4.1 Schema Overview 40
5.4.2 Entry Types 42
5.4.3 Intent and Authorization Fields 42
5.4.4 Field Role Rationale 43
5.4.5 Illustrative Derived Entry 43
5.5 Architectural Assumptions and Limits 44
5.5.1 Assumptions 45
5.5.2 Limits 45

6 Prototype Implementation 47
6.1 Prototype Scope and Boundary 47
6.2 Implementation Artifact and Project Structure 47
6.3 Prototype Inputs and Representation 48
6.4 Prototype Execution 50
6.4.1 Execution Flow 50
6.4.2 Global Assumptions and Omitted Elements 51
6.5 Family-Specific Checks 51
6.5.1 ParallelRoute 51
6.5.2 ExceptionEscape 53
6.5.3 StateSpaceGap 55
6.6 Scenario Set and Verification Basis 56

7 Evaluation and Results 59
7.1 Evaluation Purpose and Scope 59
7.2 Prototype Execution Evidence and Fixture Consistency 59
7.2.1 Comparison method and scope 59
7.2.2 Manual decision basis for expected fixture records 60
7.2.3 Worked example: p1 derivation 61
7.2.4 Execution evidence across scenario groups 63
7.3 Sensitivity and Limitation Analysis 65
7.3.1 Authentication strength only comparison in P1 65

7.3.2 Cell generation from explicit values 65
7.4 Evaluation Validity and Limitations 65
7.4.1 Internal validity 65
7.4.2 Construct validity 66
7.4.3 External validity 66
7.4.4 Conclusion validity 67

8 Conclusion and Future Work **68**
8.1 Answer to the Research Question 68
8.2 Limitations 69
8.3 Future Work 69
8.4 Final Remarks 70

AI Assistance Disclosure **71**

List of Figures **74**

List of Tables **75**

Glossary of Terms and Abbreviations

Table 1: Glossary of terms and abbreviations

Term	Meaning
Mechanism families	
`P1` / `ParallelRoute`	Mechanism family where surrounding policy structure creates a weaker alternative route to the protected outcome while the baseline does not apply to that route.
`P2` / `ExceptionEscape`	Mechanism family where exclusion structures across policies jointly leave a subset outside effective enforcement.
`P3` / `StateSpaceGap`	Mechanism family where policy conditions divide the relevant state space and leave a reachable region uncovered.
Architectural constructs	
`IntentSpec`	Declarative specification of the governance intent associated with a specific baseline policy.
`GovernedScope`	Identity, resource, and condition space declared by `IntentSpec` as the domain of baseline governance.
`ActivationBoundary`	Subset of `GovernedScope` declared as the intended active coverage region for `StateSpaceGap` analysis.
`ExceptionSpec`	An optional `IntentSpec` field declaring which exclusion cases are authorized by the baseline design. It provides the reference used to interpret an `ExceptionEscape` candidate as an intended exception or as an exclusion structure not justified by the declared intent. It is interpretive only and is not used in derivation.
Governance baseline	Declared identity governance reference used to evaluate whether the current tenant configuration preserves the intended security control. It provides the reference point for baseline artifact comparison and for semantic interpretation of surrounding policy composition.
Syntactic engine	Evaluation component that compares the current baseline policy artifact with a stored baseline snapshot.
Semantic engine	Evaluation component that evaluates whether surrounding policy composition preserves the intended governance of the baseline.

Term	Meaning
Surrounding relevant policy set	Surrounding Conditional Access policies whose resource target intersects the baseline's resource target.
Filtered working policy set	Surrounding relevant policy set after removal of policies that reference excluded condition dimensions.
Analyzable policy set	Filtered working policy set together with the baseline, used where coverage reasoning requires the baseline's contribution.
Inclusion-only scope	Scope reached by a policy's inclusion conditions before exclusions are applied.
Actual applicability	Scope that remains after exclusions are applied to a policy's inclusion-only scope.
Coverage cell	A region of the condition space formed from the possible combinations of identity, target application, network location, device platform, client application type, and authentication flow. The space is partitioned along policy and intent boundaries so that, within one cell, policy applicability and intent membership remain stable. The semantic engine applies each family check at cell level rather than separately for every possible access request.
`WitnessCell`	The specific coverage cell on which a derived entry is based. It records the evaluated combination of condition values, for example a privileged administrator accessing `AdminPortal` from a trusted location on a Windows device through a browser. This makes the finding locatable, explainable, and suitable for later re-examination.
`CellBoundaryTrace`	A record of the boundary provenance of a coverage cell. For an uncovered cell, it records which policy and intent boundaries define the cell, which surrounding policies contribute to the boundary structure, and which condition dimensions differ between the uncovered cell and the nearby policy coverage regions. This allows a `StateSpaceGap` observation to show how the uncovered region is shaped by surrounding policy structure.
Output and schema terms	
Derived entry	Structured output record produced for a specific finding by the semantic screening procedure.

Term	Meaning
Family status record	Output record indicating whether a mechanism family was attempted and, where relevant, why it did not produce an entry.
Validation path marker	Record emitted by the prototype to make preparation and family precondition paths observable when no derived entry is produced.
`interference-candidate`	`RecordType` used for candidate interaction records produced by `ParallelRoute` and `ExceptionEscape`.
`structural-observation`	`RecordType` used for `StateSpaceGap` entries that record an ungoverned cell within `ActivationBoundary`.
Candidate interaction condition	Conceptual output of `ParallelRoute` or `ExceptionEscape`, recorded in a derived entry of `RecordType interference-candidate`.
Structural coverage observation	Conceptual output of `StateSpaceGap`, recorded in a derived entry of `RecordType structural-observation`.
Retained candidate	Candidate interaction condition carried forward as a derived entry after the relevant family checks; for `ParallelRoute`, this means no discharge condition applied.
Discharged candidate	`ParallelRoute` candidate context where the strength comparison required by the abstraction model could not be validly completed.
Semantic scope exclusion	Outcome category recorded when a baseline references an excluded condition dimension and is therefore scoped out of semantic evaluation at entry.
Precondition failure	Outcome category recorded when a required input such as the `IntentSpec` binding or `GovernedScope` is missing or unresolvable.
Family not attempted	Outcome category recorded when a mechanism family is not entered, typically because its structural precondition is not met.
`attempted_no_qualifying_cells`	Family status indicating that the evaluator was entered but no evaluation cell satisfied the conditions needed to produce an entry.

Term	Meaning
`InclusionOnlyCoverage`	Derived entry field listing the policies that covered the cell before exclusions were applied; distinct from the inclusion-only scope concept.

1 Introduction

1.1 Motivation and Problem Statement

Over the past decade, cloud architectures have shifted security away from network perimeters and toward identity systems. Access is no longer controlled primarily through static segmentation. It flows through authentication decisions, authorization logic, and conditions tied to identity context. The practical consequence is that an organization's security posture now depends less on network topology and more on how identity configuration is structured and maintained over time.

Cloud identity platforms rarely operate in stable environments. Administrative adjustments, temporary exceptions, and shifting business requirements continuously reshape configuration state. Any single change may seem entirely reasonable in isolation. The problem is cumulative: over time, these adjustments alter the effective security semantics of a tenant in ways that no longer reflect what was originally intended. Governing identity in this setting requires more than reviewing individual policy artifacts in isolation. Identity must be treated as a control plane whose security properties depend on how configuration state evolves over time and how interacting controls shape effective enforcement as the environment changes.

The security relevance of this problem lies in the exposure created when effective enforcement no longer matches the declared baseline intent. Such conditions can leave access governed by less assurance than the baseline requires, even when the baseline artifact itself is unchanged. This thesis does not model how an attacker would exploit such a condition. It examines whether the condition exists within the represented policy state.

This creates a narrower research gap: existing baseline comparison can identify whether an identity policy object has changed, but it does not determine whether the surrounding policy structure still preserves the governance intent associated with that baseline. This thesis addresses that gap through semantic screening of Conditional Access policy interactions.

In this thesis, semantic screening means evaluating whether the surrounding Conditional Access policy structure can cause access to be governed differently from the enforcement intent declared by the baseline. Rather than producing only a compliant or non-compliant verdict, the screening process records the type of interaction or coverage condition that is observed within the admitted abstraction boundary. The remainder of this chapter defines the scope and research questions that follow from this framing.

1.2 Scope and Research Questions

This thesis focuses on the semantic evaluation of surrounding `Conditional Access` policy structure for declared baselines. The central concern is whether policy interactions and coverage conditions around a baseline can affect the preservation of the declared governance intent. The prototype evaluates this problem through controlled scenario packages and an explicit abstraction over policy structure, declared baseline intent, and admitted condition dimensions.

The main research question is:

> How can a semantic engine for Microsoft Entra ID Conditional Access baselines evaluate surrounding policy structure and produce structured observations within a defined abstraction boundary?

This question concerns outcomes that cannot be established by inspecting the baseline policy artifact on its own.

The question is broken down into four subquestions:

1. What abstraction is needed to represent declared baseline intent, surrounding Conditional Access policy structure, and admitted condition dimensions for semantic screening?
2. How can a bounded taxonomy distinguish policy interaction and coverage outcomes within the declared baseline scope?
3. How can a semantic screening procedure preserve distinct outcome categories instead of reducing policy interaction and coverage outcomes to a single verdict?
4. To what extent can the proposed semantic screening architecture be realized as an executable prototype that preserves the distinct outcome categories defined by the screening procedure across controlled scenarios?

1.3 Thesis Contributions

This thesis provides an architecture and prototype for semantic screening of Microsoft Entra ID Conditional Access baselines within a defined abstraction boundary. The contributions are as follows:

1. It defines an explicit abstraction for representing the information needed for semantic screening. This abstraction relates a baseline policy artifact, the surrounding Conditional Access policy structure, declared baseline intent, and the condition dimensions admitted by the analysis.
2. It introduces a bounded taxonomy for classifying policy interaction and coverage outcomes within the declared baseline scope. The taxonomy separates interaction mecha-

nisms, such as parallel routes and exception escape conditions, from structural coverage observations, such as ungoverned cells within the admitted abstraction boundary. This provides the screening procedure with distinct categories for reporting semantic outcomes.

3. It defines a semantic screening procedure that derives typed records for policy interaction and coverage outcomes. The procedure separates candidate generation from candidate interpretation, so that some candidates may be retained, others may be discharged, and structural observations can be reported without being treated as compliance failures. This allows the result to preserve the meaning of each outcome category rather than collapsing the analysis into a single compliant or non-compliant verdict.

4. It realizes the semantic screening architecture as an executable prototype and evaluates it over controlled scenario packages. The evaluation shows that the prototype produces the expected typed records across the controlled scenarios and preserves the intended distinction between retained candidates, discharged candidates, and structural observations. This supports the internal consistency of the proposed abstraction, taxonomy, and screening procedure within the stated abstraction boundary, while leaving live tenant validation and full Microsoft Entra ID platform fidelity outside the evaluation.

These contributions are limited to semantic screening within the stated abstraction boundary. The prototype evaluates the proposed abstraction, taxonomy, and screening procedure over controlled scenario packages. Experiments over live Microsoft Entra ID tenant data, full platform fidelity, and direct syntactic comparison of baseline artifacts are outside the scope of this thesis.

2 Background

Cloud identity platforms provide authentication, authorization, and policy-based access control for distributed systems. In environments such as Microsoft Entra ID, access decisions are evaluated against user identities, contextual attributes, and policy conditions rather than enforced solely at a network boundary.

Identity configuration is expressed through multiple interacting elements, including `Conditional Access` policies, `Authentication Strength`, `Role Assignments`, and `Privileged Access` controls. These elements are managed through administrative portals, APIs, and automation workflows, and together define how access is evaluated across the tenant at a given point in time.

2.1 Identity as a Control Plane in Cloud Architectures

In this thesis, a control plane refers to the part of a system that defines, evaluates, and governs access decisions. It is distinct from the data path that carries application traffic or user data. In cloud identity architectures, this control function is expressed through authentication services, role assignments, `Conditional Access` policies, and contextual evaluation rules. These mechanisms determine which principals may interact with which resources, under which conditions, and with which assurance requirements. Identity therefore acts as a governance layer for access across the tenant, because security decisions are produced from the composition of identity objects, policy conditions, and declared exceptions rather than from a single network boundary.

This gives identity systems a structural role in security governance. Ghadge et al. [9] show that structured identity governance practices are central to keeping access rights aligned with organizational objectives. In practice, this means that identity policies define not only who can access a resource, but also under what conditions and with what level of assurance.

The operational difficulty is that identity controls are administered as separate objects, while their security effect is evaluated across the tenant. This makes complexity and misconfiguration in identity and access management persistent governance problems [22]. The compositional consequences of this separation are discussed in Section 2.4.

2.2 Identity Governance in Microsoft Entra ID

This thesis examines that general problem through Microsoft Entra ID, which serves as the authentication and authorization backbone for Microsoft 365 and Azure-based services. It is a suitable setting because the tenant's effective identity posture does not reside in any single object. It emerges from the interaction of multiple control elements, including `Conditional Access` policies, `Authentication Strength` definitions, `Role Assignments`, `Privileged Identity Management (PIM)` configurations, and `Emergency Access` accounts.

Some of these control elements contribute directly to enforcement decisions, while others define exceptions and constraints on how those decisions apply. `Conditional Access` policies specify the contextual conditions under which access is granted or blocked. `Authentication Strength` definitions constrain which authentication methods are acceptable when such requirements are enforced. `Role Assignments` define which permissions principals hold, while `PIM` governs how privileged roles are activated and managed over time. `Emergency Access` accounts play a distinct role by introducing explicit exclusion from standard controls, establishing conditions under which normal enforcement requirements are bypassed. The effective security posture of the tenant therefore depends not only on whether each element is configured correctly in isolation, but also on how these elements interact and how exceptions to enforcement are defined (Figure 1).

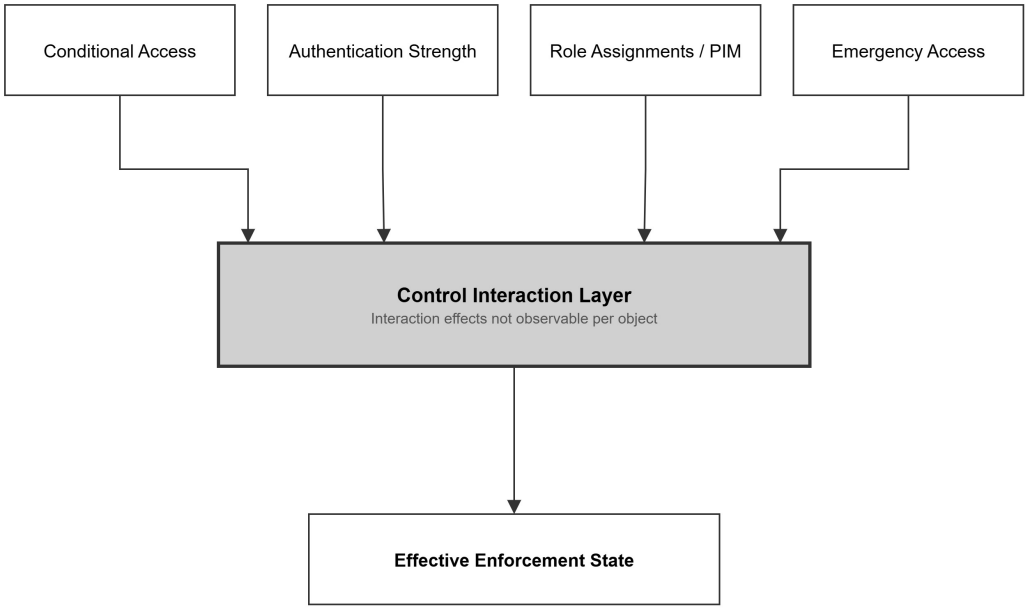

Figure 1: Compositional structure of identity enforcement in Microsoft Entra ID

Figure 1 illustrates the broader compositional setting from which the thesis problem is derived. The upper boxes represent identity control elements that are usually configured and reviewed as separate objects. The interaction layer represents the fact that their security effect is not determined only by each object in isolation, but by how the objects combine across the tenant. The resulting effective enforcement state is therefore a tenant level property. This distinction matters because the thesis studies whether the surrounding identity control structure preserves the governance role assigned to a baseline, rather than only whether the baseline object is internally valid.

2.3 Declarative Configuration as Security State

Identity controls in cloud platforms are expressed declaratively. Conditional Access rules, role assignments, and authentication method configurations describe intended constraints on behavior rather than procedural steps to execute. In this sense, the tenant's security posture is represented through configuration state: a structured description of what the control plane is supposed to enforce at a given point in time.

This thesis distinguishes configuration state from a governance baseline. Configuration state refers to the current tenant configuration that exists at a given point in time. A governance baseline refers to the explicit reference posture against which that current state is interpreted. The former describes what is currently configured; the latter describes what security properties the tenant is expected to preserve.

In an identity governance context, this requires an explicit reference representation of intended posture. NIST OSCAL [18] provides a machine readable schema for expressing compliance controls and assessment artifacts in a portable, tool agnostic form. That is valuable at the level of general control representation, but it does not by itself capture the platform specific interaction semantics through which identity controls produce effective enforcement in a tenant. For the purposes of this thesis, the relevant construct is therefore a governance baseline: an explicit reference representation of the security requirements and structural assumptions that the represented tenant state is evaluated against.

This idea has a precedent in infrastructure management. Infrastructure as Code frameworks treat configuration files as authoritative state definitions and use them as references for deployed resources. The relevant lesson for identity governance is not the deployment mechanism itself, but the need for an explicit reference state against which current configuration can be interpreted.

The same logic applies to identity systems. Once identity configuration is treated as state, it becomes possible to reason about whether that state still satisfies defined security expectations. The broader implication is that identity configuration should not be treated only as administrative configuration. It represents security state that can be analyzed, compared, and tested against an explicit reference posture.

2.4 Compositional Interaction and Drift in Identity Systems

The previous sections establish identity configuration as tenant security state. This section focuses on two ways that state can become misleading for governance: interaction between controls and drift over time.

Interaction problems can arise even when the relevant controls are all part of the intended configuration. Conditional Access policies, role assignments, exclusions, privileged activation rules, and emergency access arrangements may each appear justified when reviewed inde-

pendently, yet their combination can undermine the security guarantees they were meant to support. The issue is therefore compositional: the security meaning of a control depends not only on its own definition, but on how it interacts with other controls across the tenant.

For example, a baseline policy may remain unchanged while another policy targets the same protected application under a different set of conditions. The second policy does not override the baseline, but it can still affect the enforcement posture if it governs access in a context where the baseline does not apply. Similarly, exclusions across policies may leave part of an intended population outside effective enforcement. In such cases, the relevant question is not only whether the baseline object changed, but whether the surrounding policy set still preserves the enforcement intent associated with it.

Drift is one important source of this broader problem. Temporary exceptions may remain active after they were meant to expire, migration era policies may continue to apply after their original purpose has passed, and urgent changes may be introduced outside ordinary review. Identity tenants are also modified through several overlapping channels, including management portals, scripts, APIs, and automated workflows. Under these conditions, effective enforcement can shift without a single clearly invalid configuration object.

The result is that problematic enforcement does not require a single invalid object. The form of semantic change considered in this thesis is a shift in enforcement meaning caused by overlaps, weakened assumptions, or unintended paths in the surrounding control set.

Taken together, these interaction patterns show why baseline evaluation cannot be limited to object level change. The relevant question for this thesis is whether the surrounding `Conditional Access` policy structure still preserves the enforcement intent associated with a declared baseline. This shifts the evaluation problem from detecting isolated configuration differences to analyzing whether policy composition has altered the effective enforcement associated with that baseline.

3 Literature Review

Cloud security research increasingly treats security posture as a property that changes with the system being governed. This shift is especially important in identity platforms, where permissions, policies, and authentication controls are repeatedly modified and evaluated together through tenant operations. In this setting, posture cannot be inferred from isolated rules alone. It must be understood as a property of the effective tenant state.

This review examines four strands of related work: representation and evaluation of state, compositional exposure analysis, declarative compliance pipelines, and identity conformance. The purpose is not to summarize each work separately, but to identify the gap addressed by this thesis. In this thesis, declared baseline intent refers to the enforcement effect that a baseline control is intended to guarantee.

3.1 State-Based Evaluation and Drift Detection

A central claim in cloud and identity security research is that security cannot be assessed only by inspecting isolated policy statements. It requires a representation of the system state in which permissions, policies, and control relationships can be evaluated together.

Graph-based misconfiguration detection makes this point explicit. Van Ede et al. [8] model AWS IAM policies as a graph and identify misconfigurations as anomalies in the resulting permission structure. This is useful because it treats misconfiguration as a property of the access control topology rather than as a single rule violation. However, the method is not designed around declared baseline intent. It can identify unusual or risky structures, but its unit of analysis is permission topology rather than the enforcement effect guaranteed by a specific baseline control.

Formal verification provides a stronger mechanism for reasoning about state. Ambit [13] encodes Azure RBAC and checks whether proposed changes violate stated security boundaries. This gives a precise answer to a narrow operational question: whether a change would move the RBAC state outside a specified invariant. The limitation for this thesis is that the analysis remains centered on RBAC invariants and proposed changes. It does not address Conditional Access policy structure, exception handling, authentication strength requirements, or interaction between identity policies in an evolving tenant.

Drift detection in declarative infrastructure management reinforces the importance of comparing intended and observed state. NSync [26] observes that drift appears through cloud API activity and attempts to reconcile out of band changes back into Infrastructure as Code. This is valuable because it recognizes that drift often results from multiple control channels rather than from repository changes alone. However, the model remains focused on cloud resources and declarative infrastructure state. It reconciles resource state, not identity policy semantics. Static review methods also struggle to keep pace with change in live cloud environments [24],

which reinforces the need for state aware evaluation.

Taken together, these works show why security evaluation needs a state model. Their shared limit is that the state model is not anchored in declared baseline intent.

3.2 Compositional Exposure and Attack Path Reasoning

A second strand of related work studies how security exposure can emerge from combinations of individually valid configuration elements.

Model checking work makes this point visible by showing how exposure can emerge from composition. Shevrin and Margalit [21] encode IAM configurations and search for privilege escalation chains. Their relevance here is not the specific attack model, but the demonstration that security exposure is not local. A configuration element that appears acceptable in isolation may contribute to an unsafe reachable state when evaluated together with other permissions or controls.

A practical example of this reasoning appears in Principal Mapper (PMapper), which searches AWS IAM configurations for indirect capability chains rather than only direct entitlements [19]. Its relevance is conceptual: a principal may not appear dangerous when inspected locally, yet still be able to reach sensitive resources through a sequence of individually valid permissions and delegated actions. This supports the broader move from isolated policy inspection toward analysis of the effective authorization structure.

The limitation for this thesis is that attack path reasoning asks what an actor can reach, while this thesis asks whether policy composition changes the enforcement effect associated with declared baseline intent.

3.3 Declarative Security and Compliance as Code

Compliance as code and declarative security pipelines show how security intent can be encoded as machine readable artifacts and evaluated systematically through automation.

The broader Everything as Code paradigm [23] reflects this shift by treating infrastructure, compliance artifacts, and regulatory evidence as versioned objects managed through development workflows. Haverinen et al. [10] similarly show the value of expressing compliance knowledge in a structured, machine processable format, independent of a specific toolchain. Agarwal et al. [1] demonstrate how this idea can be implemented at production scale using NIST OSCAL and Trestle to connect machine readable compliance requirements with heterogeneous validation tools. These works are relevant because they show that governance intent can be represented explicitly. Their limitation for this thesis is that controls are represented as independent requirements, without a model of one policy modifying another policy's enforcement effect.

Yanagawa et al. [25] treat security policies as testable artifacts that can be evaluated systematically against compliance objectives. Policy as code engines such as Open Policy Agent (OPA) [7] generalize this idea by decoupling policy logic from enforcement points and evaluating rules against input state. These approaches support the view that policy logic can be formalized and executed. However, their usual operating context differs from the problem addressed here. They evaluate policies or requests against bounded inputs, while the problem here depends on interaction across a represented tenant state.

Bruns and Huth [6] address a related issue in policy composition by using a four valued logic in which conflict is treated as an explicit semantic result rather than collapsed into a binary allow or deny answer. This is relevant because the present thesis also treats composed policy behavior as something richer than direct object comparison or checklist validation.

The gap for this thesis is therefore not how to encode controls declaratively or how to run compliance checks in a pipeline. These approaches support explicit control representation, but not Conditional Access interaction semantics at the baseline level.

3.4 Identity Conformance: Protocol Correctness to Tenant Governance

In identity systems, conformance is not a single evaluation problem but consists of two distinct concerns:

- **Protocol-level** conformance is about whether authentication and authorization flows follow specifications and known security best practices. The Micro Id Gym extension in [5] fits this category. It brings conformance testing into continuous pipelines and validates OAuth and OpenID Connect deployments. That is valuable, but it targets correctness of flows, not the governance posture of a tenant.
- **Tenant-level** conformance is about whether the effective configuration preserves declared baseline intent.

A particularly relevant step in this direction is the formal modeling of the AWS authorization engine by Barnett et al. [4], who explicitly argue that many practical security questions cannot be answered by reasoning about one policy at a time. Their IAM-MultiPolicyAnalyzer is designed to verify properties across multiple policies of different types and therefore addresses interaction as a first class analysis problem. This is important for the present thesis because it provides a close precedent for moving beyond single artifact correctness toward multi-policy semantic reasoning. At the same time, the scope remains tied to AWS authorization semantics and property verification within that ecosystem. The Microsoft Entra ID problem addressed here is different. The question is not only whether a composed policy set satisfies a specified property, but how Conditional Access interaction affects the enforcement effect associated with declared baseline intent.

Formal verification offers a principled mechanism for establishing configuration conformance

against specified properties. The technical feasibility of this direction has already been demonstrated in large-scale cloud environments: Backes et al. [3] formalise the AWS policy language as SMT formulas and show that sound, automated reasoning over access control semantics is viable at production scale, with millions of invocations daily across AWS services. The limitation is structural: the analysis operates at the level of individual policy specifications and does not model the broader identity topology through which permissions propagate, such as principal hierarchies, group membership inheritance, or dynamic role activation in Microsoft Entra ID. Formal verification work such as Ambit [13] closes part of that gap by encoding Azure RBAC state and verifying whether proposed changes violate stated security boundaries. Multistep IAM analysis [21] further demonstrates how tenant configuration can enable reachability chains that are invisible to local checks. Yet neither line of work, on its own, captures tenant-level identity policy interaction in a modern SaaS environment.

In platforms like Microsoft Entra ID, the governance failures relevant to this thesis are usually not protocol violations. They are configuration-level interactions, such as Conditional Access exclusions that weaken intended coverage or authentication strength settings that fall short of a declared baseline. These arise from policy interaction, not protocol errors.

The current Microsoft ecosystem reflects this gap clearly. The Conditional Access What If tool [16] can simulate a sign in and explain which policies apply together in that specific scenario, while Maester extends this operationally through automated policy checks and validation workflows. Microsoft's more recent Conditional Access Optimization Agent broadens the analysis further by scanning for uncovered users or applications, overlapping or consolidatable policies, and exception patterns that may create gaps in enforcement. [17, 14] These tools are useful, but they remain operational rather than formally evaluative. What If is scenario bound, Maester is check driven, and the Optimization Agent is recommendation oriented. None of them provides a semantic screening procedure for evaluating whether Conditional Access policy interaction violates declared baseline intent. Microsoft further notes that What If does not test Conditional Access service dependencies, which reinforces the point that tenant-level conformance cannot be reduced to scenario-based simulation alone. [16]

These tools clarify the operational state of `Conditional Access`, but they do not provide the declared baseline intent model used in this thesis. This distinction is synthesized in the next section.

3.5 Identity Baseline Governance Gap

The reviewed literature shows a consistent separation between technical correctness analysis and semantic baseline evaluation. State-based methods provide models of configuration state, but are not anchored in declared baseline intent. Compositional approaches expose reachability and attack paths, but their endpoint is what an actor can reach. Declarative compliance pipelines make controls machine-readable, but often treat them as independent

requirements. Identity conformance work validates protocol behavior or operational policy state, but does not provide a tenant-level model for Conditional Access policy interaction.

A baseline artifact can remain syntactically valid while surrounding `Conditional Access` policies change the enforcement effect it was intended to guarantee. Individually valid policies, exclusions, authentication requirements, and target assignments can combine to change that effect in ways that object-level validity, scenario simulation, and checklist validation do not reveal.

The gap addressed by this thesis is therefore not general identity governance, misconfiguration detection, attack path analysis, or temporal drift detection. It is the problem of evaluating whether `Conditional Access` policy composition preserves declared baseline intent in the represented tenant state.

3.6 Positioning of This Thesis

Based on this gap, this thesis focuses on semantic baseline evaluation: whether a composed `Conditional Access` policy set preserves declared baseline intent.

To address this problem, the thesis introduces a bounded abstraction for representing a baseline policy artifact, declared baseline intent, surrounding `Conditional Access` policy structure, and admitted condition dimensions. It then defines a taxonomy and screening procedure for deriving structured records from policy interaction and coverage conditions. The prototype implements this procedure and evaluates it over controlled scenario packages.

The contribution is therefore not a general identity governance platform, an attack path model, or a production ready Microsoft Entra ID assessment tool. It is a bounded and inspectable method for semantic baseline evaluation in composed `Conditional Access` policy sets.

4 Methodology

This chapter explains the methodological basis of the thesis. It positions the work as a design science research study, defines how the research question is addressed through artifact design and controlled evaluation, and states the scope and limitations of the chosen method. The use of structured scenarios rather than live tenant data bounds the evaluation claims to the modeled conditions under which the prototype is exercised.

4.1 Research Paradigm and Rationale

This thesis follows the Design Science Research paradigm as proposed by Hevner et al. [11]. The approach is appropriate because the research problem is addressed by designing and evaluating an artifact rather than by observing an existing system. The artifact consists of a semantic screening architecture and an executable prototype for Microsoft Entra ID Conditional Access baseline evaluation.

The research problem concerns whether surrounding Conditional Access policy structure can alter the governance meaning of a baseline without changing the baseline artifact itself. The thesis therefore studies how declared baseline intent and surrounding policy structure can be represented and evaluated within a bounded model.

The evaluation is therefore scenario-based rather than tenant-based. This keeps the artifact evaluation aligned with the selected semantic mechanisms and limits the resulting claims to the modeled scope.

Positioning Within the DSR Cycles

The relevance cycle connects the artifact to the practical problem of identity governance in cloud environments, where effective enforcement depends on interacting policy objects rather than on a single configuration item.

The rigor cycle is provided by prior work on state-based security evaluation, declarative security, compliance as code, and identity conformance. These works motivate the need for explicit state representation and systematic evaluation, while the thesis narrows that problem to semantic screening of Conditional Access baseline intent.

The design cycle is reflected in the development of the semantic evaluation artifact, its implementation as an executable prototype, and its assessment against selected scenario outcomes.

Justification of Controlled Scenario Scope

The thesis uses controlled scenario packages rather than live enterprise tenant data. This choice follows from the purpose of the evaluation. The goal is not to measure the frequency of policy failures in real tenants, but to examine whether the proposed artifact can expose selected policy interaction outcomes under known conditions.

Controlled scenarios make the expected outcome basis inspectable and reduce the influence of historical configuration noise, undocumented exceptions, and operational context. This allows the prototype output to be compared with explicit expected records while keeping the evaluated mechanisms visible.

4.2 Research Design and Alignment

The main research question and its parts are stated in Section 1.2. This chapter does not repeat them in full. Methodologically, the questions are addressed through connected design and evaluation activities: defining a bounded representation of the problem, specifying the semantic screening logic, implementing selected parts of that logic in a prototype, and evaluating the resulting behavior under controlled conditions.

The architecture chapter defines the design. The implementation chapter describes the executable realization of the selected semantic checks. The evaluation chapter reports the observed fixture records, execution evidence, and validity limits.

4.3 Scope, Assumptions, and Boundaries

The thesis is scoped to semantic screening of Microsoft Entra ID Conditional Access baselines. The prototype does not connect to a live tenant and does not implement the syntactic comparison engine. Its evaluation is limited to structured scenario state.

For scope control, the model is limited to a fixed set of policy and access context dimensions. The dimensions included in the abstraction and those excluded from it are defined in Table 2 in Chapter 5.

The model assumes that group membership, named locations, and other tenant facts needed for the admitted dimensions have already been resolved before evaluation. The prototype therefore evaluates semantic logic over the represented facts, rather than resolving or reproducing the full operational behavior of Microsoft Entra ID.

The thesis does not evaluate RBAC role assignments, Privileged Identity Management, Entitlement Management, cross tenant access, Exchange, SharePoint, or live remediation workflows. These are outside the scope of the present artifact.

4.4 Evaluation Strategy

The evaluation strategy is fixture level consistency checking. The prototype is executed over the controlled scenarios introduced in Chapter 6, and the observed fixture records are compared with the expected records for those scenarios.

The comparison is limited to the observable record categories exposed by the prototype. It therefore evaluates whether the implemented artifact preserves the selected semantic distinctions targeted by the scenario set, not whether the taxonomy is complete or whether the approach is correct for live tenants.

The expected records are used as scenario oracles. They are part of the same research artifact as the prototype and therefore do not provide independent ground truth. The implications of this oracle dependence are analyzed in Section 7.4.1.

4.5 Methodological Limitations

The main methodological limitation is that the evaluation is based on controlled scenarios and artifact internal expected records. This supports consistency checking under known conditions, but not independent proof of semantic correctness or evidence about operational prevalence in live tenants.

A second limitation is that the evaluation inherits the abstraction boundary of the architecture. The results apply only to the represented Conditional Access dimensions and policy structures. These limitations are analyzed in Chapter 7.

5 Architecture of the Proposed Framework

The architecture is built around three entities: the baseline policy, the security expectation declared for it, and the surrounding `Conditional Access` policies. The surrounding policies are included because policies that target the same application can affect the enforcement posture through different assignments, exclusions, locations, device platforms, or client application conditions.

The difficulty is that surrounding policies can change what is effectively enforced without changing the baseline object. A second policy may allow access to the same application in a context where the baseline does not apply and impose a lower assurance requirement. Several policies may appear to cover the relevant access context before exclusions are applied, while their exclusions collectively leave a subset uncovered. Policies may also divide the condition space so that part of the intended coverage area has no applicable policy at all. These cases motivate the three interaction families used in this chapter: `ParallelRoute`, `ExceptionEscape`, and `StateSpaceGap`.

Chapter 3 established the structural governance gap that motivates this architecture. Existing approaches can assess object-level validity, operational policy state, scenario-specific applicability, or broader compositional exposure. However, they do not provide a tenant-level method for evaluating whether `Conditional Access` policy composition preserves declared baseline intent in the represented tenant state. This chapter addresses that gap by adding semantic screening to baseline evaluation. Artifact comparison remains necessary for determining whether the baseline policy artifact has changed. The semantic part of the architecture addresses the separate question of whether the surrounding `Conditional Access` policy set still preserves the enforcement effect that the baseline was intended to guarantee. The architecture therefore contains a `syntactic engine` for artifact comparison and a `semantic engine` for that screening procedure.

The `syntactic engine` takes two inputs: the current baseline policy artifact and a stored baseline snapshot. It compares them and determines whether the baseline policy artifact matches the stored baseline snapshot. The `semantic engine` takes a different set of inputs: a baseline policy artifact, its associated `IntentSpec`, and the tenant `Conditional Access` policy set. The `semantic engine` produces structured semantic results on whether the surrounding policy set continues to support the intended governance of the baseline.

Both engines run on every baseline. The syntactic engine is not gated by semantic considerations, and the semantic engine is not gated by syntactic findings. Their outputs are complementary, not cross-validating. A syntactic result showing no change in the baseline artifact does not imply a clean semantic result, because the `semantic engine` may still detect a governance failure caused by changes elsewhere in the policy set. The two engines answer different questions, so their outputs are not expected to agree, and they do not need to.

Figure 2 provides a high level view of the overall baseline evaluation architecture, showing the distinct inputs and outputs of the syntactic and semantic engines.

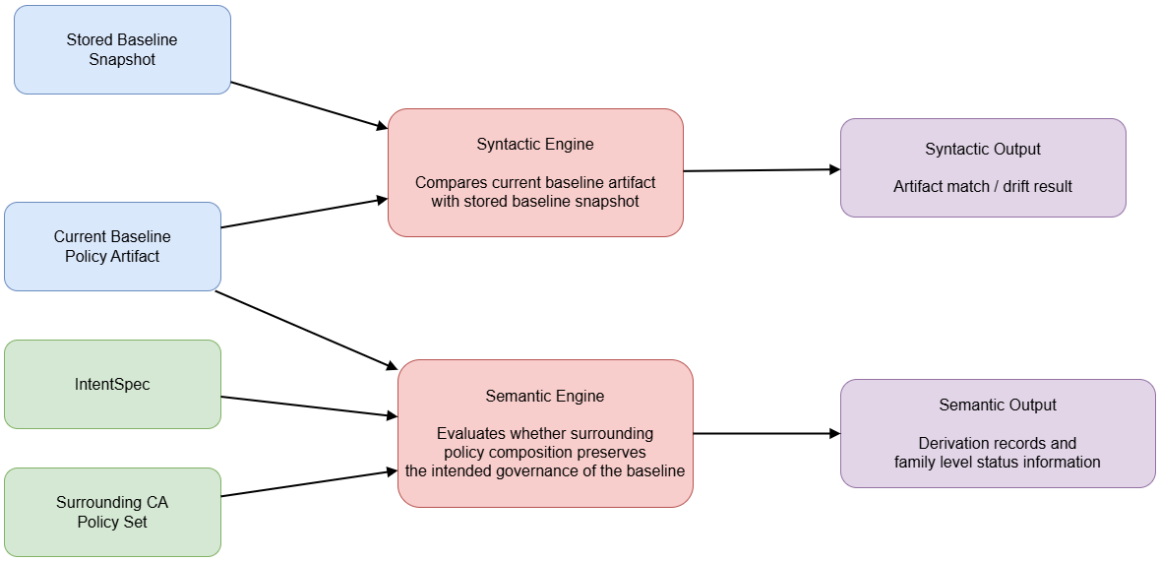

Figure 2: High level architecture of baseline evaluation

This chapter specifies the `semantic engine` because the chapter's design contribution lies in modeling and evaluating governance failures that arise from surrounding policy composition. The `syntactic engine` performs direct artifact comparison and therefore does not require comparable architectural elaboration.

The semantic engine is specified within a bounded model. It evaluates a tenant snapshot under the documented evaluation semantics of `Conditional Access` and depends on an explicit association between each baseline and its `IntentSpec`. Its findings are therefore valid only within the analyzable domain defined by that model. They are not claims about full tenant behavior outside the represented dimensions and assumptions.

The design is also limited to relationships represented in `Conditional Access` policy structure and declared baseline intent. It does not attempt to model externally generated risk or compliance signals, undocumented platform behavior, or equivalence between separate application targets that expose the same underlying system. The full set of assumptions and limits is stated in Section 5.5.

The subsections that follow specify the `semantic engine` through four design components. The `Semantic Interaction Taxonomy` defines the interaction mechanisms the engine recognizes. The `Abstraction Model for Semantic Evaluation` defines how the system state is represented for analysis and the assumptions behind that representation. The `Semantic Screening Procedure` translates those assumptions into a structured derivation sequence. The `Derived Entry Schema` defines the structure of the derived entries produced by the screening procedure.

5.1 Semantic Interaction Taxonomy

The `semantic engine` evaluates whether surrounding policy composition undermines the intended governance of a baseline. To structure this analysis, the architecture adopts a small taxonomy of interaction mechanisms. The taxonomy defines the mechanism families the engine can detect, while screening determines which of them, if any, are instantiated for a given baseline.

The taxonomy consumes the `IntentSpec` defined in Section 5.2.1. In this subsection, `GovernedScope` denotes the declared governance domain of a baseline, spanning identity, resource, and condition dimensions. Its full representation is specified in Section 5.2.3. The three mechanism families use the intent fields differently according to their analytical requirements.

The taxonomy is presented under two design commitments that remain fixed throughout this chapter:

- Every baseline undergoes semantic screening; there is no prior gate that routes baselines away from the semantic engine before screening begins.
- Scope exclusion is determined syntactically using the excluded dimensions defined in Table 2. The reason follows from the abstraction boundary defined in Section 5.2.

Additional declarations required for specific mechanisms are introduced where those mechanisms are defined.

The taxonomy is intentionally narrow. It does not aim to provide an exhaustive taxonomy of all possible governance failures in `Conditional Access`. Instead, it defines a bounded set of mechanisms selected to capture policy-composition failures that are not detectable from the baseline artifact alone. Three mechanism families are included: `ParallelRoute` (P1), `ExceptionEscape` (P2), and `StateSpaceGap` (P3). The taxonomy excludes architecturally distinct phenomena, including failures that depend on external service semantics, undocumented platform behavior, or defects arising within a single policy rather than from policy interaction.

The mechanism families are not mutually exclusive. The taxonomy classifies failure mechanisms rather than baselines, so a single baseline may instantiate candidate interaction conditions under more than one family. The semantic engine therefore permits multiple candidates for the same baseline, supported by the available evidence.

The three mechanism families differ in the epistemic status of the outputs they produce. `ParallelRoute` and `ExceptionEscape` generate interference candidates that may require later confirmation, contextual interpretation, or follow on analysis, whereas `StateSpaceGap` generates structural observations whose significance is not fixed by derivation alone. These distinctions are specified in Section 5.4.2.

ParallelRoute. `ParallelRoute` captures the case where surrounding policy composition governs a different access route to the same protected outcome, or an equivalent governance outcome, under weaker conditions than the baseline requires. The baseline remains syntactically intact, and no single policy appears independently unsafe. The governance concern emerges only when the relevant policies are evaluated together, making clear that the protected outcome remains reachable through an alternative branch with lower effective requirements.

For example, suppose the declared intent is that privileged administrative access to a sensitive application should require phishing resistant authentication. The baseline policy enforces this requirement for the primary administrative access path. A surrounding policy also targets the same application in a context where the baseline does not apply, but it imposes only a weaker authentication requirement. The baseline artifact may remain unchanged, yet the protected application is still reachable through a parallel route with lower assurance than the baseline intent requires.

This family therefore requires that at least one surrounding policy apply to a context within the `GovernedScope` where the baseline does not apply. Where the baseline and surrounding policies are co-applicable, the effective outcome is interpreted using the grant and block semantics defined under *Grant or block requirement*. A `ParallelRoute` condition is therefore searched outside baseline applicability, while still within the declared `GovernedScope`.

A concrete derivation of a `ParallelRoute` candidate is presented in Section 7.2.3.

ExceptionEscape. `ExceptionEscape` captures the case where exclusion structures across multiple policies jointly produce a subset within the `GovernedScope`. That subset is excluded from every policy that would otherwise apply to it. As a result, no policy that would otherwise contribute to preserving the baseline's intended governance remains applicable. The critical property is not the presence of an exclusion in one policy, but the composition of exclusions across the policy set. The escape appears only when those exclusion structures are evaluated together.

For example, suppose two policies both target the administrative application and both include privileged administrators before exclusions are applied. One policy excludes administrators in a temporary migration group, while another excludes administrators using browser client applications from trusted network locations. An administrative access attempt by a user in the temporary migration group, using a browser from a trusted network location, remains inside the declared `GovernedScope`. However, after exclusions are applied, neither policy remains applicable to that access context. The uncovered subset is therefore produced by the combined exclusion structure rather than by a single policy in isolation.

StateSpaceGap. `StateSpaceGap` captures the case where two or more policies divide a governance-relevant condition space in a way that leaves part of that space uncovered. The mechanism arises when a reachable combination of condition values falls outside the applicability of every relevant policy. Within the present architecture, this family is limited to gaps that arise from policy composition within the analyzable `Conditional Access` state. An uncovered combination qualifies as a `StateSpaceGap` observation only where the `IntentSpec`, through a declared `ActivationBoundary`, provides a basis for treating that combination as part of the intended coverage region rather than as an intended design boundary.

For example, suppose the declared `ActivationBoundary` states that privileged administrative access should be covered across a defined set of locations and device platforms. One policy covers Windows devices from trusted locations, while another policy covers macOS devices from untrusted locations. A remaining combination, such as macOS access from a trusted location, lies inside the declared `ActivationBoundary` but outside the applicability of every relevant policy. The issue is not an exclusion from an otherwise applicable policy, but an uncovered cell produced by the way policy conditions divide the state space.

5.2 Abstraction Model for Semantic Evaluation

Semantic evaluation cannot operate directly over the full complexity of `Conditional Access` policy behavior. The abstraction model defines a minimal representation that makes this space tractable by capturing only the dimensions necessary to detect governance failures caused by interactions among `Conditional Access` policies. The abstraction operates over three inputs jointly: the baseline policy artifact, the surrounding `Conditional Access` policy set as configured, and the associated `IntentSpec`. Without policy structure, interference cannot be located. Without intent, a structural deviation cannot be identified as a governance failure. The abstraction therefore does not represent the `Conditional Access` policy layer alone. It represents that policy layer as interpreted through declared baseline intent.

5.2.1 Intent Representation

IntentSpec as an architectural input. `IntentSpec` is the declarative specification of the governance intent associated with a specific baseline policy. It is a first class architectural input to the semantic engine, because the baseline artifact alone does not determine whether an uncovered region reflects an intended design boundary or an unaccounted coverage failure. The baseline artifact records configuration, not the design intent behind it, and the same configuration can be consistent with multiple intents. `IntentSpec` therefore supplies the declared intent fields required for semantic evaluation, notably `GovernedScope`, and, where declared, `ActivationBoundary` and `ExceptionSpec`. These fields are expressed using the same condition dimensions assumed by the abstraction model, so they can be interpreted directly within the semantic engine. Each baseline is assumed to be associated with an `IntentSpec` through an unambiguous binding relation. This association is an architectural precondition of

semantic evaluation.

`GovernedScope`. `GovernedScope` denotes the declared governance domain of the baseline. It provides the common scope within which semantic findings are evaluated. Its full representation, including why it is not reducible to identity alone, is specified in Section 5.2.3.

`ActivationBoundary`. Where declared in `IntentSpec`, `ActivationBoundary` denotes the portion of `GovernedScope` that the baseline is intended to cover. It does not define governance intent independently of `GovernedScope`; it restricts the intended coverage region within it. This distinction matters for structural gap analysis, because an uncovered cell may indicate a semantic failure only if it falls within the declared `ActivationBoundary`. Where no `ActivationBoundary` is declared, this distinction cannot be made.

`ExceptionSpec`. Where declared in `IntentSpec`, `ExceptionSpec` denotes the authorization basis for exclusion structures that may arise during `ExceptionEscape` analysis. It does not define coverage scope. Its role is interpretive rather than generative: derivation may identify an exclusion structured candidate without `ExceptionSpec`, but later interpretation of that candidate requires the authorization reference that `ExceptionSpec` provides.

Figure 3 illustrates the conceptual relation between the baseline policy artifact and its associated `IntentSpec`, including the roles of `GovernedScope`, `ActivationBoundary`, and `ExceptionSpec`.

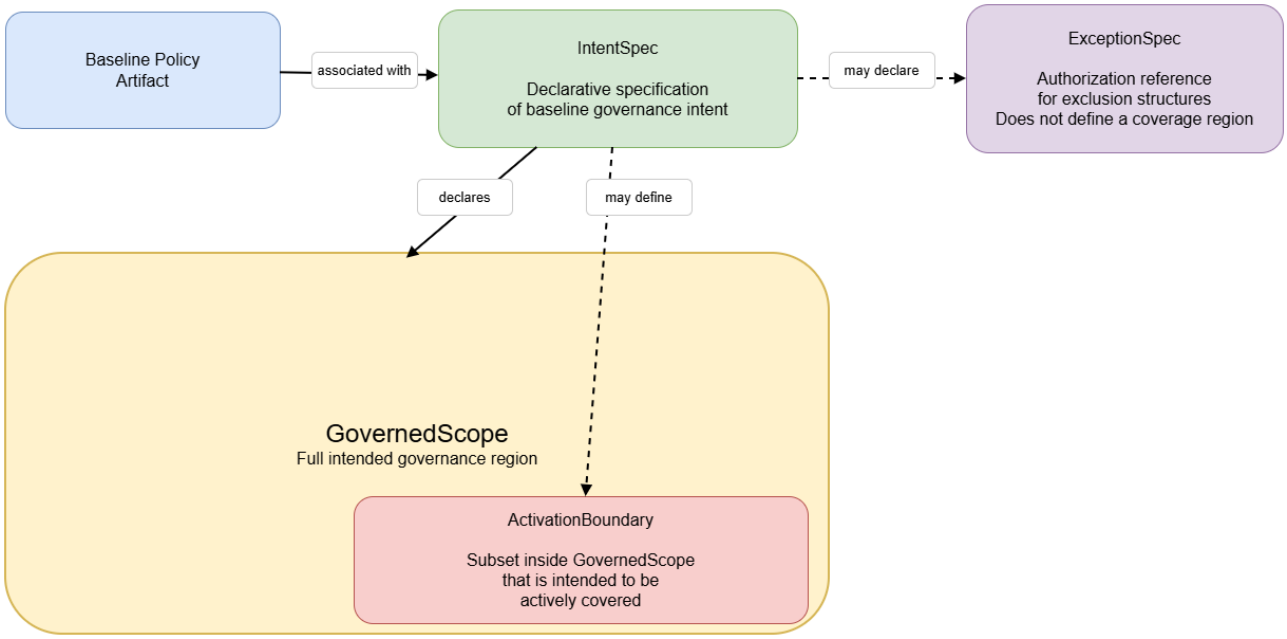

Figure 3: Conceptual relation between the baseline policy artifact and its associated `IntentSpec`

5.2.2 Analytical Boundary and Dimensions

The abstraction restricts semantic analysis to dimensions that can be represented from three input categories. `Conditional Access` policy structure denotes fields declared in policy objects, including assignment conditions, resource targets, exclusions, grant requirements, and block conditions. Resolved tenant state denotes tenant facts that are not contained directly in a policy object but can be resolved at evaluation time, such as group membership, user attribute values, and named location definitions. Request context denotes values associated with an access attempt, such as the target application, network location, device platform, client application type, and authentication flow branch.

Table 2 summarizes the condition and scope dimensions included in the semantic abstraction and the externally generated signal dimensions excluded by the present model.

Table 2: Included and excluded dimensions in the semantic abstraction

Dimension	Representation or reason
Included dimensions represented in the semantic abstraction	
Identity and principal membership	Identity assignments in policy structure, with group membership or user attribute values resolved from tenant state
Target resource or application	Request target matched against resource target fields declared in `Conditional Access` policies
Network location	Request network location matched against named location definitions resolved from tenant state
Device platform	Request device platform matched against device platform conditions in policy structure
Client application type	Request client application type matched against client application conditions in policy structure
Authentication flow branch	Request authentication flow branch matched against authentication flow conditions in policy structure
Excluded dimensions outside the represented tenant snapshot	
Sign in risk	Externally generated risk signal
User risk	Externally generated risk signal
Device compliance state	Externally generated compliance signal

The excluded signal dimensions fall outside the present abstraction because they are generated

by services beyond the represented `Conditional Access` policy layer or are not derivable from the tenant snapshot used by the semantic engine. Policies conditioned on these signals therefore fall outside the scope of the present semantic engine. This is a boundary choice of the model, not a claim that such baselines are semantically unanalyzable in principle.

The table concerns condition and scope dimensions. It does not list every represented policy field. Inclusions, exclusions, grant requirements, block conditions, authentication strength requirements, and resource target fields are defined in the policy representation subsection below. In the table, target resource or application denotes the access target dimension that is matched against those resource target fields. Broader exclusions, such as undocumented platform behavior, external service semantics, temporal dynamics, and platform behavior not represented in declared policy fields, are architectural limits rather than dimensions and are therefore stated in Section 5.5.

5.2.3 Policy Representation and Governed Scope

What the abstraction represents. Each `Conditional Access` policy is represented through two modeling components:

- *Applicability scope.* The first component is the policy's applicability scope. It is determined by combining policy configuration, resolved tenant state, and request context dimensions across identities, target applications, network locations, device platform types, client application types, and authentication flow branches. In the present model, this dimension is represented in two forms: inclusion only scope, which denotes the scope reached by a policy's inclusion conditions before exclusions are applied, and actual applicability, which denotes the scope that remains after exclusions are applied.
- *Grant or block requirement.* The second component is the policy's grant or block requirement: the condition that must be satisfied for access to be granted within the applicability scope, or the block condition that denies access regardless of other applicable policies. Under the documented `Conditional Access` evaluation semantics, a block condition in any applicable policy results in denial for that scope. For grant conditions, all applicable grant requirements must be satisfied, so the effective requirement is interpreted conjunctively [15]. Grant requirements are not uniformly comparable across control types. Authentication strength requirements can be compared by assurance level because the `Conditional Access` platform provides a platform-defined strength ordering among them. In the present model, a requirement A is weaker than a requirement B when every request satisfying B also satisfies A, and at least one request satisfies A without satisfying B. Controls such as device join type or approved client application are not naturally orderable against authentication strength requirements. The abstraction therefore supports comparison only within a common control type for which the platform defines such a strength ordering.

Governed scope. The baseline's governed scope is declared through the `GovernedScope` field of `IntentSpec`. It bounds the access contexts that remain relevant to the baseline even where the baseline policy itself does not apply. In the present model, governed scope is represented as a slice of identity, resource, and condition space, so it is not reducible to identity alone. A baseline governing privileged users accessing administrative applications under all network conditions has a different governed scope from one governing the same identities only under specific network conditions, even if the identity set is identical. Interference mechanisms can operate on the condition dimension of governed scope, not only on the identity dimension. Reducing governed scope to identities would cause interference cases that operate on condition values to appear incorrectly as falling inside or outside scope.

All three mechanism families are analyzed relative to `IntentSpec`, but they do not consume identical intent fields because they do not impose the same analytical requirements. `ParallelRoute` and `ExceptionEscape` ask whether a candidate condition within the baseline's intended governance arises. For this purpose, `GovernedScope` provides the common domain of analysis. `StateSpaceGap` imposes a stronger requirement. It asks not only whether an uncovered cell is relevant to the baseline, but whether absence of coverage at that cell is interpretable as a structural gap rather than as an intended design limit. `ActivationBoundary`, where declared, supplies that stronger condition by identifying the subset of the governed domain that is intended to be actively covered. The family specific use of intent therefore reflects differences in analytical requirement, not the use of unrelated semantic bases.

For `ExceptionEscape`, `GovernedScope` only bounds the relevance of the candidate. The coverage expectation is not inferred from `GovernedScope` alone, but from the `inclusion-only` scope of the relevant policies: multiple policies reached the cell before exclusions were applied, while no policy remains applicable after those exclusions are evaluated.

Relevant policy set. Semantic analysis begins from the surrounding tenant `Conditional Access` policy set as the initial structural input. From that set, the abstraction constructs a relevant policy set for each baseline based on shared resource target. The relevant policy set for a baseline is all `Conditional Access` policies whose resource target set includes at least one application also present in the baseline's resource target set, under `Conditional Access` resource targeting semantics.

Security equivalence between different resource targets cannot be derived from `Conditional Access` policy structure alone without importing access semantics external to the policy layer. The abstraction therefore uses shared resource target as the structural basis for relevance. A parallel route to a functionally equivalent but structurally distinct resource target is not detectable within this abstraction.

For example, an organization may register a legacy access portal and a modern web application as separate app objects in Entra ID, even though both provide access to the same underlying data. A baseline policy governing the modern application shares no resource target with a

policy targeting the legacy portal. The abstraction cannot detect that a weaker route through the legacy portal may reach the same protected data, because the two app registrations are structurally distinct targets. Establishing that equivalence would require importing knowledge of what each application exposes, which lies outside the policy layer.

The relevant policy set is intentionally broader than applicability scope overlap. A policy whose applicability scope shares no evaluation context with the baseline may still belong to the relevant policy set if it targets the same resource. This is necessary because parallel route analysis operates in contexts where the baseline does not apply, and exclusion escape analysis may involve policies governing adjacent rather than overlapping populations. For example, a policy governing guest users may share no evaluation context with a baseline governing privileged internal accounts. If both target the same administrative application, the guest-user policy still belongs to the relevant policy set. This is because the baseline does not apply in that context, yet a weaker route through that policy may still bear on governance of the same protected resource.

5.2.4 Mechanism Specific Modeling Assumptions

Interference mechanisms. The abstraction model translates the interaction mechanisms introduced in Section 5.1 into a representation suitable for analysis. The present subsection defines the modeling assumptions under which those mechanisms can be analyzed.

ParallelRoute. Analysis of `ParallelRoute` requires the model to represent both applicability scope and grant or block outcome so that it can identify contexts within `GovernedScope` where the baseline does not apply but at least one surrounding policy does. The effective outcome in those contexts is interpreted using the grant and block semantics defined under *Grant or block requirement*.

ExceptionEscape. Analysis of `ExceptionEscape` requires the model to distinguish a policy's inclusion only scope from its actual applicability after exclusions are applied. This makes it possible to test whether a cell is collectively excluded across policies that would otherwise cover it.

StateSpaceGap. Analysis of `StateSpaceGap` requires a finite condition space, a declared `ActivationBoundary`, and a representation of policy induced boundaries that can identify uncovered cells whose geometry is supported by surrounding policy boundaries from at least two distinct mismatch dimensions.

A condition space gap and an exclusion escape may arise from the same structural situation. Collective exclusion across the relevant policy set may remove a condition combination from all applicability scopes while also leaving an ungoverned cell within the declared coverage region. In such a case both mechanisms are present, and neither is defined as the residual of the other.

5.3 Semantic Screening Procedure

This section specifies the screening procedure that the semantic engine executes against each baseline selected for evaluation. The prior sections have established the mechanism taxonomy and the abstraction model. This procedure translates those components into a structured, repeatable sequence that takes a baseline, its associated `IntentSpec`, and the surrounding `Conditional Access` policy set as inputs, and produces derivation records. These records preserve analytically distinct outcome types rather than collapsing them into a single result. Each outcome type is introduced at the step where it arises and enumerated at the end of the procedure. The procedure runs in eight steps.

At a high level, the procedure can be read as four phases. Step 0 is the entry stage, where the engine either admits the baseline to semantic derivation or records semantic scope exclusion or precondition failure. Steps 1 to 4 prepare the semantic inputs and analytical structures by extracting `IntentSpec` fields, constructing and filtering the relevant policy set, and building the fragments and coverage cells used by the family checks. Steps 5 and 6 perform family evaluation and discharge: the three mechanism families are evaluated independently, and discharge applies only to `ParallelRoute` candidates whose comparison cannot be completed under the abstraction model. Step 7 records the preserved outcome categories as structured derivation output.

Figure 4 provides an overview of the semantic screening procedure, showing the main derivation sequence, early exit paths, the `StateSpaceGap` precondition branch, the `ParallelRoute` discharge path, and the final output categories. The `ActivationBoundary` branch shown in Step 5 uses the status extracted in Step 1; it is drawn at the point where that status determines whether `StateSpaceGap` is attempted.

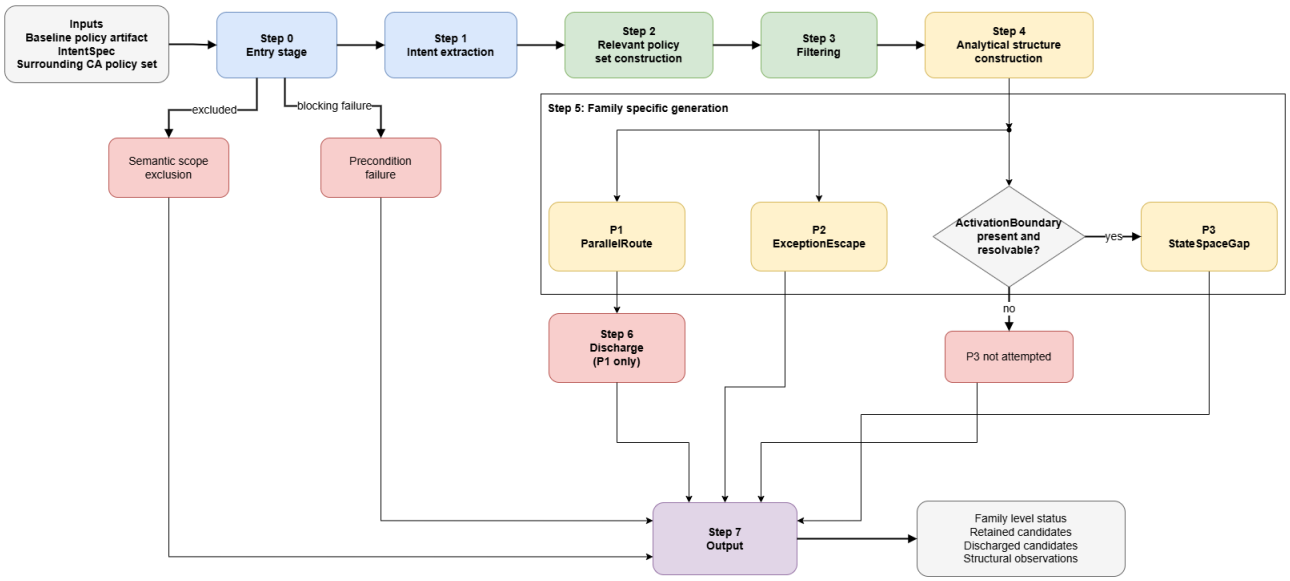

Figure 4: Overview of the semantic screening procedure and its principal outcome paths

Step 0: Entry Stage. Step 0 is the entry stage for semantic derivation. The successful path admits the baseline to the remaining screening steps. The two termination paths, semantic scope exclusion and precondition failure, each produce a distinct structured termination record.

The first termination path is semantic scope exclusion. Every baseline enters semantic screening; there is no prior classification gate. Within screening, a baseline is scoped out if its assignment conditions reference any dimension excluded by the abstraction boundary in Table 2. A baseline that references these dimensions cannot be faithfully represented within the abstraction boundary defined in Section 5.2. The exclusion is syntactically determined from the baseline's policy configuration. An excluded baseline has therefore undergone semantic screening but cannot proceed into semantic derivation. This is recorded as a semantic scope exclusion termination record identifying the baseline and the excluded dimension referenced.

The second termination path is precondition failure. Where the baseline passes the scope check, two blocking input conditions must hold before the remaining steps can run. First, a valid binding must exist between the baseline and its `IntentSpec`. Second, the `GovernedScope` field must be present in the `IntentSpec` and resolvable using condition dimensions within the abstraction boundary. If both conditions hold, the baseline is admitted to the remaining screening steps. If either condition fails, the procedure cannot continue. The specific failure is recorded as a precondition failure termination record identifying which condition was not met and what was absent or malformed.

By contrast, absence of `ActivationBoundary` or `ExceptionSpec` does not block derivation. Those omissions are handled at later family-specific steps or preserved in the derivation output where relevant.

Step 1: `IntentSpec` **Extraction.** Step 1 produces one fixed output and one conditional output from the `IntentSpec`, together with an `ActivationBoundary` status that carries forward to subsequent steps.

The fixed output is `GovernedScope`, extracted from `IntentSpec` and expressed across the condition dimensions defined in the abstraction model.

The conditional output is `ActivationBoundary`. Its status takes one of three values. If no `ActivationBoundary` is declared in the `IntentSpec`, the status is absent. If one is declared and can be fully resolved using the condition dimensions within the abstraction boundary, the status is present and resolvable, and the boundary is carried forward as an output. If one is declared but cannot be resolved within the abstraction dimensions, the status is present but unresolvable.

Step 1 also records whether `ExceptionSpec` is declared in the `IntentSpec`. This status is carried into `ExceptionEscape` entries at Step 7 to indicate whether the authorization

reference relevant to later interpretation was available at derivation time. `ExceptionSpec` is not consumed by derivation itself.

The `StateSpaceGap` family cannot be attempted under either the absent or present-but-unresolvable status. In both cases the reason is recorded at Step 5 as a family precondition failure. These two statuses are recorded distinctly: absent reflects an `ActivationBoundary` omission; present but unresolvable reflects a resolvability failure given the abstraction boundary.

Step 2: Relevant Policy Set Construction. Using the relevance relation defined under *Relevant policy set*, the engine constructs the surrounding relevant policy set for the baseline from the surrounding tenant `Conditional Access` policy set. Selection is based on shared resource targeting under `Conditional Access` resource targeting semantics, not on applicability scope overlap. The baseline itself is not included in this surrounding relevant policy set. It is added later to the analyzable policy set where coverage reasoning requires its contribution, so that the roles of the baseline and the surrounding policies remain distinguishable.

Step 3: Filtering by Abstraction Boundary. Before any analytical structure is constructed, policies in the surrounding relevant policy set whose assignment conditions reference any dimension excluded by the abstraction boundary in Table 2 are removed. A policy is removed if any part of its assignment conditions references such a dimension.

The result is the filtered working policy set. The analyzable policy set is the filtered working policy set together with the baseline. It is used in Steps 4 and 5 wherever coverage reasoning requires the baseline's contribution to be included.

Step 4: Analytical Structure Construction. Step 4 constructs the analytical structures required for the family-specific checks in Step 5. Two structures are produced: policy fragments and coverage cells.

Policy fragments. For each policy in the filtered working policy set and for the baseline, the procedure constructs a fragment that records the policy's applicability before and after exclusions are applied. Each fragment records four fields: the inclusion-only scope; the actual applicability scope; the intersection of the actual applicability scope with `GovernedScope`; and the declared exclusion predicates over `GovernedScope`. The inclusion-only scope is retained explicitly because `ExceptionEscape` analysis depends on the distinction between coverage before and after exclusion application.

Coverage cells. The condition space is partitioned into coverage cells so that later family-specific checks can be performed over uniform regions. The partition is constructed from the policy boundaries relevant to analysis together with the boundaries of `GovernedScope` and, where Step 1 yields a present-and-resolvable result, `ActivationBoundary`. Each cell is therefore entirely within or entirely outside every boundary relevant to subsequent checks.

For each cell, the procedure records four classifications:

- whether the baseline applies within the cell under actual applicability;
- which surrounding policies apply within the cell under actual applicability;
- which surrounding policies include the cell before exclusions are applied;
- whether the baseline includes the cell before exclusions are applied.

Together, these classifications preserve both actual applicability and inclusion only coverage, which are required by the later mechanism checks.

For each cell, a `CellBoundaryTrace` is also recorded. This trace preserves the boundary provenance needed for later structural analysis, including whether a boundary segment arises from policy conditions, `GovernedScope`, or `ActivationBoundary`. The family-specific checks in Step 5 operate over the analyzable domain established through these structures.

Step 5: Family-Specific Candidate Generation. The three mechanism families defined in Section 5.1 are applied independently to the coverage cells. `ParallelRoute` and `ExceptionEscape` generate candidate interaction conditions. `StateSpaceGap` generates structural coverage observations. For each family, the outcome remains distinct: the family may be recorded as not attempted, recorded as attempted with no qualifying cells, or recorded as producing one or more entries. Where screening produces neither candidate interaction conditions nor structural observations across the assessed mechanism families, this absence is recorded as a modeled result, not as a claim that the baseline is structurally independent of surrounding policy composition.

ParallelRoute (P1). For `ParallelRoute`, the engine evaluates cells inside `GovernedScope` where the baseline does not apply. The analysis operates at the level of the effective outcome produced by all jointly applicable surrounding policies, not at the level of an individual surrounding policy considered in isolation. A cell qualifies for this family only if at least one surrounding policy applies after exclusions. If no surrounding policy applies, the cell does not produce a candidate. For each qualifying cell, the engine applies the grant and block semantics defined under *Grant or block requirement* to determine the effective surrounding outcome. Cells with a surrounding block outcome are not retained as weaker access routes. Grant outcomes are compared only where the supported comparison model permits reduction to a comparable requirement. A retained candidate is generated only where the surrounding effective grant requirement is weaker than the baseline requirement associated with the declared intent. If no qualifying cells are found, the family records that no qualifying cells were observed. If a qualifying cell is found but the grant comparison cannot be supported by the model, comparison failure is handled as described in Step 6. After Step 0, this family is attempted without an additional family specific precondition.

ExceptionEscape (P2). Coverage reasoning for `ExceptionEscape` uses the analyzable policy

set. The analysis considers cells within `GovernedScope` where no policy in the analyzable policy set applies under actual applicability, including the baseline. If the baseline applies at a cell, that cell remains governed within the analyzable domain and does not qualify for this mechanism.

A multi-policy guard is then applied using each policy's inclusion-only scope, which represents the set of cells each policy reaches before its exclusions are applied. If fewer than two policies satisfy this condition, the absence of actual applicability does not reflect collective exclusion action within the analyzable domain and falls outside the scope of `ExceptionEscape`.

An `ExceptionEscape` candidate interaction condition is generated where at least two policies include a cell before exclusions are applied, no policy in the analyzable policy set applies there after exclusions are taken into account, and the loss of applicability is explained by declared exclusions across those policies. If no cell within `GovernedScope` satisfies these conditions, `ExceptionEscape` is attempted with no qualifying cells found.

StateSpaceGap (P3). `StateSpaceGap` generates structural coverage observations rather than candidate interaction conditions. A structural coverage observation records that a cell satisfies geometric conditions within the policy environment represented by the analyzable policy set. Whether such an observation reflects a genuine governance interaction is not established by derivation alone.

The family is attempted only where Step 1 yields a present-and-resolvable `ActivationBoundary`. Otherwise, `StateSpaceGap` is not attempted.

Coverage reasoning for `StateSpaceGap` first uses the analyzable policy set to determine whether a cell is uncovered. The analysis considers cells within `ActivationBoundary` where no policy in the analyzable policy set applies under actual applicability. This uncoveredness test includes the baseline, because a cell covered by the baseline is not a gap.

For such uncovered cells, the boundary support test is narrower. The baseline is not counted as a qualifying boundary support contribution. Qualifying support is derived from policies in the filtered working policy set, that is, policies other than the baseline that remain after abstraction boundary filtering. A policy contribution qualifies as boundary support only when the policy does not include the uncovered cell and differs from that cell in exactly one admitted inclusion dimension.

The required structural guard is therefore twofold. First, the uncovered cell must have qualifying support from at least two distinct policies other than the baseline. Second, those support contributions must cover at least two distinct mismatch dimensions. This establishes a guarded surrounding policy boundary condition. It does not by itself establish that the absence of coverage is causally produced by policy interaction.

A `StateSpaceGap` structural coverage observation is generated where an uncovered cell within `ActivationBoundary` satisfies this guarded support condition. If no cell within `ActivationBoundary`

satisfies these conditions, `StateSpaceGap` is attempted with no qualifying cells found.

The distinction between `ExceptionEscape` and `StateSpaceGap` rests on the guard applied to the ungoverned cell. `ExceptionEscape` requires collective exclusion across policies that included the cell before exclusions were applied. `StateSpaceGap` requires a guarded surrounding policy boundary condition with support from at least two distinct mismatch dimensions, and does not depend on any specific exclusion explanation.

Step 6: Candidate Discharge. Step 6 applies only to `ParallelRoute` (P1) candidate interaction conditions. `ExceptionEscape` (P2) candidate interaction conditions are not subject to discharge, because their derivation does not depend on strength comparison within a comparable control type. `StateSpaceGap` (P3) produces `structural-observation` entries and is therefore outside the scope of discharge.

One discharge class is recognized: comparison failure. This discharge class arises where `ParallelRoute` analysis has identified a candidate comparison context but cannot validly complete the strength comparison required by the abstraction model. Three sub-cases are distinguished. The first is lack of comparability between the baseline's declared grant requirement and the effective surrounding grant outcome. The second is an effective surrounding outcome that is composite across more than one control type and cannot be reduced to a single comparable requirement. The third is a baseline grant structure that is itself composite or otherwise not reducible to a single comparable requirement. Where none of these conditions holds, the candidate is retained.

Where one of these conditions holds, the candidate interaction condition is discharged. The derivation record preserves the discharged candidate together with its discharge class and sub-case. This preserves the distinction between a family that was attempted but produced no qualifying cells, a family that generated a candidate that could not survive admissible comparison, and a family that generated a retained entry.

Step 7: Derivation Record Output. The derivation output preserves the following six distinct top-level outcome categories introduced across the procedure:

- semantic scope exclusion;
- precondition failure;
- mechanism family not attempted;
- discharged candidate interaction condition;
- retained candidate interaction condition;
- structural coverage observation.

The output also preserves family-level status information, so that each mechanism family remains distinguishable as not attempted, attempted with no qualifying cells found, or productive of one or more entries.

The derivation procedure emits entry records of two schema-defined types, corresponding to interference-candidate outcomes and structural-observation outcomes. Their detailed structure, field specifications, and interpretive significance are defined in Section 5.4. The relation between top-level termination records, family-level status records, and derived entries is summarized in Figure 5.

5.4 Derived Entry Schema

The schema defines the structure of an individual derived entry. An entry is a structured artifact produced for a specific finding at Step 7 and carried forward as part of the semantic engine output.

Figure 5 situates derived entries within the full derivation output. If screening terminates at Step 0, the output consists of either a semantic scope exclusion record or a precondition failure record, and no family-level status records or derived entries are produced. If Step 0 is passed, the output consists of family-level status records together with zero or more derived entries. The family-level status records indicate whether each mechanism family was not attempted, attempted with no qualifying cells found, or productive of one or more entries. Where a family was not attempted, the status record also preserves the reason for non-attempt. These status records are not entries and are therefore outside the scope of this schema.

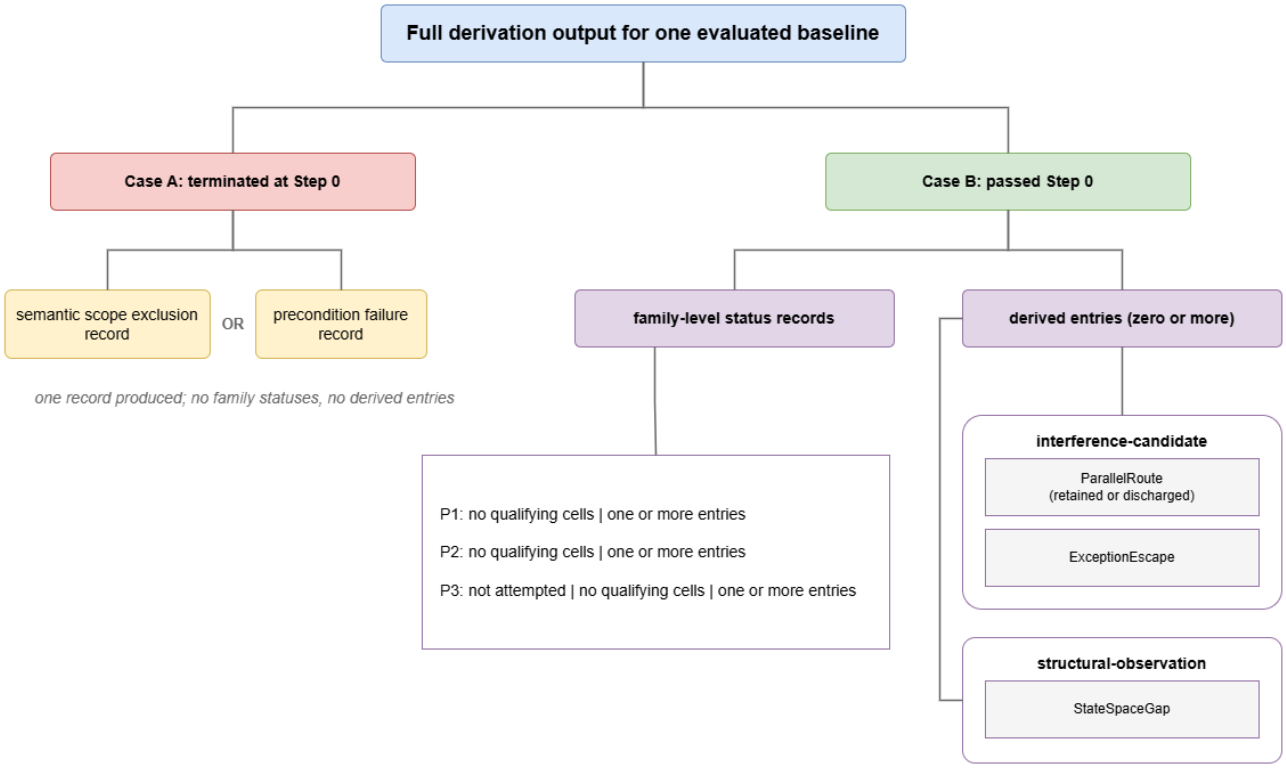

Figure 5: Full derivation output structure for one evaluated baseline

A single baseline may produce multiple derived entries. Each entry carries its own record type, field set, and derivation status independently.

5.4.1 Schema Overview

The schema defines a derived entry as a structured record whose populated fields depend on the mechanism family that produced it. The schema uses two labels to describe field roles. A field is *interpretation-critical* when its absence makes the meaning or later use of the entry indeterminate for that entry type. For example, `ActivationBoundary` status is interpretation-critical for a `StateSpaceGap` entry because the observation cannot be interpreted without knowing the declared activation region in which the uncovered cell was evaluated. A field is *derivation-traceable* when it preserves derivation time evidence needed for audit, re-derivation, or later interpretation, without itself determining the interpretive significance of the entry. For example, `TriggerCondition` is derivation-traceable because it records why the entry was produced, but the entry type and its interpretive meaning do not depend on that field alone.

Table 3 summarizes which fields are populated for each mechanism family and whether each populated field is interpretation-critical or derivation-traceable. The table concerns derived entries only. It does not define Step 0 termination records or family-level status records.

Table 3: Derived entry fields and roles by mechanism family

Field	ParallelRoute	ExceptionEscape	StateSpaceGap
Entry identity fields			
RecordType	IC	IC	IC
MechanismFamily	IC	IC	IC
WitnessCell	IC	IC	IC
Intent and context fields			
GovernedScope	DT	IC	DT
ActivationBoundary status	–	–	IC
ExceptionSpec	–	IC	–
IntentFieldsConsumed	DT	DT	DT
Evidence and pattern fields			
TriggerCondition	DT	DT	DT
CandidatePattern	DT	DT	–
ObservationPattern	–	–	DT
Coverage and trace fields			
InclusionOnlyCoverage	–	DT	–
CellBoundaryTrace	–	–	IC
StructurallyImplicatedPolicies	DT	DT	DT
Outcome fields			
Status	DT	–	–
DischargeClass	DT when discharged	–	–
DischargeReason	DT when discharged	–	–

Note. IC means *interpretation-critical*. DT means *derivation-traceable*. A dash indicates that the field is not populated for that mechanism family.

Several schema fields in Table 3 require brief clarification. `MechanismFamily` identifies the family that produced the entry, while `WitnessCell` records the concrete evaluated cell on which the entry is based. `CandidatePattern` records the derived interference pattern for `ParallelRoute` and `ExceptionEscape`, whereas `ObservationPattern` records the structural pattern observed for `StateSpaceGap`. `StructurallyImplicatedPolicies` records the policies that are directly involved in the derivation, not the full relevant policy set. `IntentFieldsConsumed` records which declared intent fields were used in producing or interpreting the entry. `Status` records the retained or discharged state of a `ParallelRoute` candidate after the discharge step.

5.4.2 Entry Types

The schema defines two entry types, distinguished by the `RecordType` field. `RecordType` takes one of two values: `interference-candidate` for `ParallelRoute` and `ExceptionEscape` entries, and `structural-observation` for `StateSpaceGap` entries.

An `interference-candidate` carries a derivation-time assertion about a structural condition in the policy configuration. That assertion may later be confirmed, contextualized, or interpreted outside the derivation procedure, but the entry itself records only what derivation established. A `structural-observation` carries a derivation-time assertion about an ungoverned cell within `ActivationBoundary`. It does not itself assert a governance violation. The interpretive range of a `structural-observation` entry is therefore constrained by the epistemic status of the carried assertion.

5.4.3 Intent and Authorization Fields

`IntentSpec`-derived fields are carried into derived entries where they are needed to preserve the meaning of what derivation established.

`GovernedScope` is carried in every derived entry as a reference field. Witness cells are derived relative to it, and the field preserves the scope within which the entry was generated.

`ActivationBoundary` status is carried in `StateSpaceGap` entries only. It records whether the activation region was present and resolvable at derivation time. Non-attempt of the `StateSpaceGap` family due to `ActivationBoundary` precondition failure is recorded at the family-status level, not as an entry.

`ExceptionSpec` is carried for `ExceptionEscape` entries as the authorization reference relevant to later interpretation of the carried collective exclusion structure. The schema establishes its role as contextual support for the derived entry; the formal authorization judgment does not belong to the schema itself.

Intent related incompleteness is not uniform in effect. Missing or unresolvable `GovernedScope`, or invalid binding between the baseline and its `IntentSpec`, terminates derivation at Step 0 and therefore does not appear inside an entry. Entry level incompleteness refers only to non-blocking fields whose absence does not prevent derivation but constrains later interpretation. In the present schema, this includes missing `ExceptionSpec` for `ExceptionEscape` entries. For `StateSpaceGap`, missing or unresolvable `ActivationBoundary` is recorded at family status level as non-attempt rather than as an entry.

5.4.4 Field Role Rationale

The role assignments in Table 3 reflect what each entry type needs in order to remain independently interpretable. `CellBoundaryTrace` and `ActivationBoundary` status are interpretation-critical for `StateSpaceGap` because the structural observation depends on boundary provenance and on the declared activation region in which the uncovered cell was evaluated. `ExceptionSpec` is interpretation-critical for `ExceptionEscape` because the authorization basis of the collective exclusion structure cannot be interpreted from policy structure alone. `GovernedScope` is interpretation-critical for `ExceptionEscape` because the candidate is meaningful only as a collective exclusion within the declared governance scope. For `ParallelRoute` and `StateSpaceGap`, `GovernedScope` preserves the derivation boundary, but the main interpretive basis is carried by the witness cell, candidate or observation pattern, and the family specific fields. It is therefore derivation-traceable for those entry types.

5.4.5 Illustrative Derived Entry

To make the schema concrete, this subsection uses a simplified `ParallelRoute` scenario. The associated `IntentSpec` declares privileged administrator access to `AdminPortal` as the relevant `GovernedScope`. The baseline policy, `CA_BL_Admin_PhishingResistant`, requires phishing resistant authentication, but it does not apply when the access attempt comes from a trusted location. A surrounding policy, `CA_SUR_Admin_TrustedMFA`, also targets `AdminPortal` and applies to privileged administrator access from trusted locations, but requires only standard MFA.

The witness cell is therefore a privileged administrator accessing `AdminPortal` from a trusted location, on a Windows device platform, using a browser client application and a standard interactive authentication flow. At this cell, the baseline does not apply, the surrounding policy does apply, and the effective surrounding grant requirement is weaker than the baseline requirement. The screening procedure therefore derives a retained `ParallelRoute` candidate.

Table 4 shows the simplified derived entry produced for this scenario. The example is illustrative rather than a complete prototype output. It shows how one entry preserves the record type, mechanism family, witness cell, derivation status, and derivation basis for later interpretation.

The baseline is part of the entry context, while `StructurallyImplicatedPolicies` records the surrounding policy structure that gives rise to the candidate. Fields used by other families or statuses, such as `InclusionOnlyCoverage`, `ObservationPattern`, `DischargeClass`, `DischargeReason`, `ActivationBoundary` status, `CellBoundaryTrace`, and `ExceptionSpec`, are omitted because they are not populated for a retained `ParallelRoute` example.

Table 4: Illustrative derived entry for a retained `ParallelRoute` candidate

Field	Role	Example value
`RecordType`	IC	`interference-candidate`
`MechanismFamily`	IC	`ParallelRoute`
`Status`	DT	`retained`
`GovernedScope`	DT	Privileged administrators accessing `AdminPortal`, as declared in the baseline `IntentSpec`
`WitnessCell`	IC	Privileged administrator; `AdminPortal`; trusted location; Windows device platform; browser client application; standard interactive authentication flow
`TriggerCondition`	DT	`CA_BL_Admin_PhishingResistant` does not apply at the witness cell because the access is from a trusted location; `CA_SUR_Admin_TrustedMFA` applies to the same application at that cell
`CandidatePattern`	DT	The baseline requires phishing resistant authentication, while the effective surrounding grant requirement at the witness cell is standard MFA
`Structurally ImplicatedPolicies`	DT	`CA_SUR_Admin_TrustedMFA`, a surrounding policy targeting `AdminPortal` and applying at the witness cell
`IntentFields Consumed`	DT	`GovernedScope`

Note. IC means *interpretation-critical*. DT means *derivation-traceable*.

Here, `retained` means that the candidate interaction condition remains in the output because the required `ParallelRoute` comparison can be completed within the abstraction model. The entry does not establish a confirmed governance violation outside the modeled policy environment. It records that the screening procedure derived a `ParallelRoute` candidate with a weaker surrounding requirement at the witness cell. Later interpretation depends on the preserved witness cell, the declared intent fields, and the surrounding policy context.

5.5 Architectural Assumptions and Limits

The architecture specified in this chapter operates under explicit assumptions and produces findings whose validity is bounded by the analyzable domain established by the abstraction model. These assumptions and limits are stated here to make the scope of the semantic engine's claims precise.

5.5.1 Assumptions

Conditional Access evaluation semantics. The model assumes the grant and block evaluation semantics stated under *Grant or block requirement*. Changes to these platform semantics would require reassessment of the mechanism definitions and their derivation logic.

Tenant snapshot sufficiency. The architecture operates over a tenant state snapshot taken at a single moment of evaluation. Policy configuration, group membership, user attributes, named location membership, and other resolved tenant state are assumed to be represented consistently within that snapshot. Dynamic changes occurring outside the snapshot boundary are not modeled.

IntentSpec binding. Each baseline is assumed to be associated with an `IntentSpec` through an unambiguous binding relation. The semantic engine takes this binding as an architectural precondition rather than constructing it. In operational use, the binding is therefore a governance artifact produced outside the engine itself.

Intent field availability. `GovernedScope` is assumed to be present and resolvable for every baseline admitted to semantic derivation. Where declared, `ActivationBoundary` and `ExceptionSpec` are assumed to be interpretable in the forms required by their respective mechanism families and the later interpretation of derived entries.

Comparable control types. Where mechanism derivation depends on comparison of grant requirements, comparison is assumed only within control types for which the abstraction model admits a valid basis of comparison. The architecture does not assume universal comparability across heterogeneous grant controls.

5.5.2 Limits

Model relative findings. The architecture is model relative by design. Its findings are claims about the analyzable domain defined by the abstraction model, not about full tenant semantics beyond that boundary.

Abstraction boundary exclusions. The abstraction excludes the signal dimensions listed in Table 2, as well as undocumented platform behavior and external service semantics. Baselines whose governance logic depends on such factors fall outside the semantic scope of the present design.

Adversarial behavior. The architecture does not model attack execution or adversary behavior. It does not reason about credential compromise, token theft, consent phishing, malicious policy modification by an insider, exploitation of application vulnerabilities, or attacker movement through the tenant. The semantic engine evaluates whether the represented policy structure preserves the declared baseline intent within the abstraction boundary. It does not establish that a derived candidate is exploitable in an operational attack scenario.

Structural resource target relevance. The architecture operates only over relationships visible in `Conditional Access` policy structure. Functionally equivalent but structurally distinct resource targets, such as separate application registrations that expose the same underlying data, are not detectable as related within this abstraction.

Temporal dynamics. The derivation procedure is snapshot based by design. Transient policy states, time dependent conditions, and changes occurring after the evaluation snapshot are not modeled.

Platform specificity. The architecture is instantiated for `Microsoft Entra ID Conditional Access` and relies on its documented evaluation semantics. Application to other identity platforms would require validation of the abstraction, the mechanism taxonomy, and the derivation logic under the semantics of those environments.

Dependency on `IntentSpec` authoring quality. The quality of semantic analysis depends on the quality of the `IntentSpec` associated with each baseline. Incomplete, inaccurate, or inconsistently authored intent declarations affect the semantic claims the engine can make. Intent authoring quality is therefore a governance dependency external to the semantic engine itself.

This chapter has specified the semantic engine through its taxonomy, abstraction model, screening procedure, derived entry schema, and the assumptions and limits under which its findings are valid. The following chapter shows how this architecture is realized in the prototype implementation.

6 Prototype Implementation

This chapter describes the implementation of the semantic engine defined in the previous chapter. The prototype executes the semantic checks over controlled scenario packages and reports fixture level results. It does not implement the syntactic engine.

6.1 Prototype Scope and Boundary

The implementation is limited to the part of the framework needed to evaluate governance failures that arise from policy interaction. Components outside that purpose, including syntactic baseline comparison and live tenant integration, are outside the prototype boundary.

The prototype is a research artifact rather than a faithful production model of Microsoft Entra ID Conditional Access. It loads structured scenario packages, constructs analyzable policy sets, generates governed cells, and emits the result types defined by the design.

A direct consequence of this boundary is that platform behavior outside the admitted abstraction is not modeled.

6.2 Implementation Artifact and Project Structure

The prototype is implemented in C# as a command line executable. At a high level, the executable resolves a scenario key, loads the corresponding fixture package, prepares the analyzable policy set, invokes the family specific evaluators, and compares the observed fixture level records with the expected records stored in the scenario package.

The repository [12] is organized around the prototype source code and a `fixtures/` directory containing the controlled scenarios.

Listing 1 shows the relevant structure schematically.

```
src/
  SemanticPrototype/
    Program.cs
    ...
fixtures/
  manifest.json
  <scenario_key>/
    baseline/
    surrounding/
    intent/
    expected.json
  ...
```

Listing 1: Schematic repository and fixture structure

The repository separates prototype logic from controlled scenario data. The `src/` directory contains the executable logic, while `fixtures/` contains scenario packages selected through `manifest.json`. A scenario key is the command line identifier used to choose one controlled run, such as a retained `ParallelRoute` case or a `StateSpaceGap` observation case.

6.3 Prototype Inputs and Representation

The prototype accepts a structured input package that contains the baseline under evaluation, a set of policy objects, and a set of intent specifications. At execution time, the baseline is identified explicitly, the policy set provides the surrounding configuration against which that baseline is assessed, and the intent specification set provides the declared semantic context required for interpretation of the baseline.

Listing 2 presents an abstract schema level view of the main structures.

```
PrototypeInput
  baseline_under_evaluation
  policies : list of PolicySpec
  intent_specs : list of IntentSpec

ConditionRegion
  user
  location
  resource
  device_platform
  client_app
  auth_flow

PolicySpec
  id
  include : ConditionRegion
  exclude : ConditionRegion
  effect : {grant, block}
  grant_controls : list?
  referenced_excluded_dimensions : list?

IntentSpec
  baseline_id
  governed_scope : ConditionRegion
  activation_boundary : ConditionRegion?
  exception_spec : object?

Cell
  user
  location
  resource
  device_platform
  client_app
  auth_flow
```

Listing 2: Abstracted input and internal data structures used by the prototype

The prototype uses simplified policy and intent structures that retain only the fields required for semantic screening. Policy objects keep inclusion and exclusion regions, effect, grant controls where relevant, and references to excluded dimensions used by the abstraction boundary checks. This is intentionally narrower than the full `Conditional Access` policy representation.

Policy conditions and intent scope declarations share the `ConditionRegion` form. A region is expressed over the six admitted dimensions shown in Listing 2, and each dimension contains either explicit named values or a wildcard. This allows policy matching, governed scope matching, activation boundary matching, and cell construction to use one common representation.

`IntentSpec` is treated as an explicit input artifact. `Cell` is constructed internally during execution and represents one concrete combination of the admitted dimensions.

6.4 Prototype Execution

This section describes the runtime sequence used by the prototype after the input representation has been established. The focus is on how the implemented components are ordered during execution and where the prototype deliberately stops short of the full architecture.

6.4.1 Execution Flow

Execution begins by selecting the baseline policy identified in the input package. The prototype then performs the initial semantic entry checks that determine whether execution may proceed. These checks include baseline lookup, baseline scope exclusion against the dimensions that lie outside the admitted abstraction, and binding of the baseline to exactly one corresponding intent specification.

After the baseline and its intent specification have been established, the prototype constructs the surrounding relevant policy set. Relevance is determined through shared resource targeting, so the prototype retains only surrounding policies whose resource scope overlaps with that of the baseline. This produces the initial surrounding policy set for the current run.

The abstraction boundary filter is applied next. Surrounding policies that reference excluded dimensions are removed before semantic checks are attempted. The remaining surrounding policies form the filtered working set, which is combined with the baseline to produce the analyzable policy set used by the family evaluators.

Once the analyzable policy set has been established, the prototype generates evaluation cells over the governed scope declared in the bound intent specification. These cells form the working evaluation space for the semantic checks.

The prototype then invokes the three family specific evaluators in turn. Their decision rules are described in Section 6.5, and their outputs constitute the semantic results used in the evaluation chapter.

6.4.2 Global Assumptions and Omitted Elements

The prototype operates on symbolic inputs that are already resolved into the forms needed by the admitted dimensions. It does not reconstruct tenant state from external sources during execution. Group expansion, named location resolution, and related state derivation therefore remain outside the prototype and are assumed to have been resolved before input is supplied.

The working evaluation domain is bounded by the explicit named values that appear in the structured inputs. Cell generation is performed over this admitted working domain rather than over a full reconstruction of all possible platform states.

Several architectural elements are not implemented as full runtime artifacts. The prototype does not emit the complete wrapper record structure described at the architectural level, but records selected preparation and family status outcomes as compact markers for scenario observation.

6.5 Family-Specific Checks

The three mechanism families are implemented as separate evaluators with distinct decision rules. The general execution flow has already been described in the previous section. The focus here is on the decision rules that distinguish the families and determine whether a result is retained, discharged, or recorded as a structural observation.

6.5.1 ParallelRoute

`ParallelRoute` is implemented as a comparison over governed cells where the baseline does not apply but at least one surrounding policy still does. The evaluator does not assess isolated surrounding policies. It assesses the effective surrounding outcome produced by all simultaneously applicable surrounding policies at a given cell.

For each governed cell, the prototype first checks whether the baseline applies. If it does, the cell is ignored for this family because no alternative route exists outside baseline applicability. If the baseline does not apply, the prototype collects the policies in the filtered working set that still apply at that cell after exclusions have been taken into account. If no surrounding policy applies, the cell does not qualify. If any applicable surrounding policy has block effect, the cell is also discarded for ParallelRoute, because the effective surrounding outcome is denial rather than a weaker grant route.

If the effective surrounding outcome remains a grant outcome, the prototype attempts strength comparison. Comparison is intentionally limited to the `authentication strength` control type, the only control type that the prototype treats as comparable. The baseline grant requirement must therefore be reducible to one comparable authentication strength value. The surrounding outcome is handled by reducing each applicable surrounding grant policy

separately. When all of them are reducible within the admitted control type, the prototype takes the maximum of the individual strengths as the effective surrounding strength at that cell.

A retained `ParallelRoute` candidate is produced when the effective surrounding strength is weaker than the baseline strength at a cell where the baseline does not apply and the surrounding route still grants access. If comparison cannot be completed because the baseline or an applicable surrounding grant policy is not reducible within the admitted comparison model, the evaluator emits a discharged `ParallelRoute` candidate. The discharge reason records whether reducibility failed in the baseline grant requirement or in a surrounding grant policy. This preserves the distinction between a non-qualifying route and a route that cannot be compared under the prototype model.

Listing 3 summarizes this decision logic in pseudocode.

```
for each cell in governed_scope_cells:

    if baseline applies at cell:
        skip cell

    applying = policies in filtered_working_set
               that apply at cell

    if applying is empty:
        skip cell

    if any policy in applying has block effect:
        skip cell

    baseline_strength, ok = reduce(baseline.grant_controls)
    if not ok:
        emit discharged candidate (baseline_not_reducible)
        continue

    surrounding_strength = -1
    for each grant policy p in applying:
        s, ok = reduce(p.grant_controls)
        if not ok:
            emit discharged candidate (surrounding_not_reducible)
            mark cell as discharged
            break
        surrounding_strength = max(surrounding_strength, s)

    if cell was discharged:
        continue

    if surrounding_strength < baseline_strength:
        emit retained candidate
```

Listing 3: ParallelRoute evaluation logic

6.5.2 ExceptionEscape

`ExceptionEscape` is implemented as a collective exclusion condition, not as a grant strength comparison. Its logic compares inclusion before exclusions with actual applicability after exclusions.

For each governed cell, the prototype identifies the policies in the analyzable policy set that

include the cell before exclusion is applied. It then separately identifies the policies that still apply after exclusion has been taken into account. A cell qualifies for `ExceptionEscape` only if at least two policies include it before exclusion and no policy in the analyzable set applies there after exclusions are applied. This ensures that the family captures a collective loss of coverage rather than a simple absence of matching policies.

The prototype records two related outputs for this family. `InclusionOnlyCoverage` records the policies that covered the cell before exclusions were applied. `StructurallyImplicatedPolicies` records the policies whose exclusion structures are implicated in the disappearance of actual applicability at that cell.

When these conditions are met, the prototype emits an `ExceptionEscape` candidate for the cell. No discharge stage is applied to this family, because the family does not depend on reducibility or strength comparison. Its output is therefore either a retained candidate or no qualifying cell.

Listing 4 summarizes the implemented `ExceptionEscape` decision logic at the same abstraction level as the other family evaluators.

```
for each cell in governed_scope_cells:

    inclusion_only_policies = policies in analyzable_policies
                              that include cell before exclusions

    applying_after_exclusions = policies in analyzable_policies
                                that apply at cell after exclusions

    if size(inclusion_only_policies) < 2:
        skip cell

    if applying_after_exclusions is not empty:
        skip cell

    structurally_implicated = policies in inclusion_only_policies
                              whose exclusion contains cell

    emit retained candidate
        with witness_cell = cell
        and  inclusion_only_coverage = inclusion_only_policies
        and  structurally_implicated_policies = structurally_implicated
```

Listing 4: ExceptionEscape evaluation logic

6.5.3 StateSpaceGap

The prototype treats `StateSpaceGap` as a structural observation rather than an interference candidate. It is evaluated over cells inside the declared activation boundary and only when an activation boundary is present in the bound `IntentSpec`. If no activation boundary is declared, the family is not attempted.

For each cell inside the activation boundary, the prototype first checks whether any policy in the analyzable policy set applies. For each uncovered cell, the prototype constructs a `CellBoundaryTrace`. The trace records the witness cell, the declared activation boundary, and policy contributions needed to determine whether surrounding policies provide boundary support.

The reported boundary support summary is derived from that trace. A contribution counts as support only when it comes from a surrounding policy in the filtered working set and differs from the witness cell in exactly one admitted inclusion dimension. Support must cover at least two distinct mismatch dimensions before the prototype emits a structural observation.

Listing 5 summarizes the implemented `StateSpaceGap` decision logic.

```
if intent_spec.activation_boundary is null:
    return not_attempted (activation_boundary_absent)

cells = build_cells_within(activation_boundary, analyzable_policies)

for each cell in cells:

    if any policy in analyzable_policies applies at cell:
        skip cell

    trace = build_cell_boundary_trace(
        cell,
        activation_boundary,
        analyzable_policies)

    support_contributions =
        contributions in trace where
            policy is in filtered_working_policies
            and policy is not baseline
            and policy does not include cell
            and policy has exactly one include mismatch dimension

    picked = group support_contributions by mismatch dimension,
             selecting the lowest policy identifier
             for each dimension

    if picked has fewer than 2 entries:
        skip cell

    emit structural observation
        with boundary_trace = trace
        and  boundary_supporting_policies = ids in picked
        and  boundary_mismatch_dimensions = dims in picked
```

Listing 5: StateSpaceGap evaluation logic

6.6 Scenario Set and Verification Basis

The prototype is exercised through fifteen synthetic scenarios. Six scenarios form the canonical set used to activate the main executable paths of the implemented semantic checks and preparation stages. Five challenge scenarios extend this set with policy noise, distributed exclusion structure, comparison discharge under unsupported controls, `StateSpaceGap` non-

attempt, and precondition failure. Three negative guard scenarios exercise cases where a family evaluator is attempted but no qualifying candidate or observation should be produced. One multi-family scenario exercises a case where the same baseline produces records under more than one mechanism family in a single fixture run.

The expected output file records fixture level expectations for each scenario. The prototype compares the observed runtime output with these expectations using compact fixture level records, rather than full architectural derivation entries. This comparison is used only to confirm that the selected implementation paths produce the expected scenario outcomes.

The scenario set is summarized in Table 5. The table keeps the concrete fixture setup visible in the thesis, while leaving result interpretation to Chapter 7.

Within the canonical scenarios listed below, `scope` and `filter` exercise abstraction boundary handling at the baseline level and at the surrounding policy level respectively.

Table 5: Scenario setup cues used by the prototype

Scenario	Setup cue	Fixture surface
Canonical scenarios		
`p1`	Baseline targets `admin_regular` access to `admin_portal`, requires `phishing_resistant_mfa`, and excludes `external`; surrounding external policies use lower assurance controls, including `mfa` and `password`.	Retained `ParallelRoute`.
`p1d`	Baseline keeps the external exclusion shape, but its grant control is `device_compliance`, which is outside the supported authentication strength comparison model.	Discharged `ParallelRoute`.
`p2`	Baseline and surrounding policies include the governed external cell before exclusions; their exclusions remove actual applicability for that cell.	Retained `ExceptionEscape`.
`p3`	Baseline covers the corporate browser branch; surrounding boundary policies cover adjacent location and client application combinations; intent declares an `ActivationBoundary` for the external browser case.	`StateSpaceGap` structural observation.
`filter`	A relevant surrounding policy references the excluded assignment dimension `device_compliance`, so it is removed by abstraction boundary filtering before family evaluation.	Filtering path marker.
`scope`	The baseline itself references the excluded assignment dimension `sign_in_risk`, so semantic screening stops at entry.	Scope exclusion path marker.

Continued on next page

Scenario	Setup cue	Fixture surface
Challenge scenarios		
`c1`	The retained `ParallelRoute` structure from `p1` is kept, while additional surrounding policies target another resource or another user as policy noise.	Retained `ParallelRoute`.
`c2`	The `ExceptionEscape` structure is kept, but the exclusion pattern is distributed across multiple surrounding policies using device based and location based exclusion variants.	Retained `ExceptionEscape`.
`c3`	Baseline uses a supported `mfa` grant requirement; the surrounding external policy uses `some_unknown_control`, which is outside the supported comparison model.	Discharged `ParallelRoute`.
`c4`	Baseline and surrounding policies follow the `StateSpaceGap` style setup, but the intent declares `GovernedScope` without `ActivationBoundary`.	`StateSpaceGap` not attempted.
`c5`	Baseline and surrounding policies follow the `ParallelRoute` style setup, but the intent artifact omits `GovernedScope`.	Precondition failure.
Negative guard scenarios		
`n1`	Baseline requires `phishing_resistant_mfa`; surrounding external policies include `phishing_resistant_mfa` and `password`, so the aggregated surrounding requirement is not weaker.	No qualifying cells.
`n2`	Baseline contains the external exclusion pattern, but the fixture has no surrounding policy files.	No qualifying cells.
`n3`	Intent declares an `ActivationBoundary`, but the surrounding set provides only one boundary support direction under the trace derived support criterion.	No qualifying cells.
Multi-family scenario		
`m1`	Baseline and one surrounding policy include the external witness cell before exclusions and neither applies after exclusions; two additional surrounding policies provide boundary support from location and client application mismatch dimensions.	Retained `ExceptionEscape` and `StateSpaceGap` observation.

The prototype source code and scenario artifacts are available in the accompanying GitHub repository [12].

7 Evaluation and Results

This chapter evaluates the proposed architecture through the prototype introduced in Chapter 6. It reports the evidence produced by controlled fixture execution and explains the limits of the resulting claims. The evaluation does not assess the prototype as a production system and does not claim full correspondence with Microsoft Entra ID platform behavior.

7.1 Evaluation Purpose and Scope

The evaluation examines whether the prototype produces the fixture records expected by the controlled scenario set and whether those records preserve the outcome distinctions defined by the architecture. The focus is on fixture record agreement, observable execution paths, and the limits of the claims that can be drawn from controlled execution.

The evaluation uses the fifteen scenario fixtures introduced in Chapter 6: six canonical scenarios, five challenge scenarios, three negative guard scenarios, and one multi-family scenario. These fixtures follow the scenario package structure defined in Section 6.6.

7.2 Prototype Execution Evidence and Fixture Consistency

This section reports the comparison method and the execution evidence observed across the scenario set. It first states the comparison method, then makes the expected record basis inspectable, traces one concrete derivation, and reports the observed records, status outcomes, and path markers.

The implemented comparison surface is narrower than the full architecture and supports fixture agreement checking rather than full architectural derivation record comparison.

For a fixed fixture package and prototype version, execution is deterministic. The reported evidence is therefore interpreted as fixture agreement for the stated prototype version.

7.2.1 Comparison method and scope

The comparison is carried out by executing each scenario and comparing the observed fixture records with the expected records stored in the scenario package. The comparison uses the compact fixture level records described in Section 6.6.

The expected records are treated as fixture level oracles for the selected scenarios. They encode the thesis interpretation of the architecture for each controlled scenario rather than independent external ground truth. The implications of this oracle dependence are analyzed in Section 7.4.1.

7.2.2 Manual decision basis for expected fixture records

To make the expected records inspectable, the evaluation treats each expected fixture record as the result of applying a decision rule defined by the architecture in Section 5.3 to a controlled scenario setup. Table 6 summarizes the decision rule basis for each expected fixture record.

Table 6: Manual decision rule basis for expected fixture records

Scenario	Decision rule basis and expected fixture record
`p1`	Baseline absent at the witness cell; weaker surrounding grant applies (Step 5, `ParallelRoute` retained). **Expected record:** retained `ParallelRoute` candidate.
`p1d`	Candidate context reached; baseline grant requirement is not reducible (Step 6, `ParallelRoute` discharge). **Expected record:** discharged `ParallelRoute` candidate.
`p2`	At least two policies include the cell before exclusions; no policy applies after exclusions (Step 5, `ExceptionEscape`). **Expected record:** retained `ExceptionEscape` candidate.
`p3`	Ungoverned cell lies inside `ActivationBoundary` and satisfies the implemented trace derived boundary support condition (Step 5, `StateSpaceGap`). **Expected record:** `StateSpaceGap` structural observation.
`filter`	Relevant surrounding policy references an excluded abstraction dimension and is removed before family evaluation (Step 3, abstraction boundary filtering). **Expected record:** relevant policy filtering path marker.
`scope`	Baseline references an excluded abstraction dimension at semantic entry (Step 0, semantic scope exclusion). **Expected record:** semantic scope exclusion path marker.
`c1`	Retained `ParallelRoute` condition remains despite additional irrelevant surrounding policies (Step 5, `ParallelRoute` retained). **Expected record:** retained `ParallelRoute` candidate.
`c2`	Distributed exclusions remove actual applicability after prior inclusion by multiple policies (Step 5, `ExceptionEscape`). **Expected record:** retained `ExceptionEscape` candidate.
`c3`	Candidate context reached; surrounding grant requirement is not reducible (Step 6, `ParallelRoute` discharge). **Expected record:** discharged `ParallelRoute` candidate.
`c4`	`StateSpaceGap` precondition fails because no `ActivationBoundary` is declared (Step 5, `StateSpaceGap` not attempted). **Expected record:** `StateSpaceGap` not attempted path marker.

Continued on next page

Scenario	Decision rule basis and expected fixture record
c5	Required GovernedScope is missing, so derivation stops at precondition checking (Step 0, precondition failure). **Expected record:** precondition failure path marker.
n1	Surrounding route exists, but the effective surrounding requirement is not weaker than the baseline (Step 5, ParallelRoute no qualifying cells). **Expected record:** ParallelRoute attempted with no qualifying cells.
n2	No surrounding policies are present, so the multi-policy inclusion guard fails (Step 5, ExceptionEscape no qualifying cells). **Expected record:** ExceptionEscape attempted with no qualifying cells.
n3	Ungoverned cell lacks the support required by the implemented trace derived condition (Step 5, StateSpaceGap no qualifying cells). **Expected record:** StateSpaceGap attempted with no qualifying cells.
m1	The witness cell is included before exclusions by POLICY_BASELINE and POLICY_EXTERNAL_EXCLUDED, but no policy applies after exclusions. The same cell lies inside ActivationBoundary and satisfies the trace derived support condition through client_app and location mismatch dimensions (Step 5, ExceptionEscape and StateSpaceGap). **Expected record:** retained ExceptionEscape candidate and StateSpaceGap structural observation.

The negative guard scenarios make selected no qualifying cells outcomes observable. They show that the implemented family guards suppress findings in the selected controlled cases.

7.2.3 Worked example: p1 derivation

This subsection applies the screening procedure of Section 5.3 to the p1 fixture as a concrete manual derivation. It does not introduce new decision rules. Its purpose is to show how the expected p1 fixture record in Table 6 follows from the architectural rules and the fixture inputs.

Inputs. The p1 fixture contains one baseline policy, two surrounding policies, and one bound IntentSpec. The baseline includes admin_regular access to admin_portal from corp_net or external, excludes external, and requires phishing_resistant_mfa. The two surrounding policies cover the external route and require mfa and password, respectively. The bound IntentSpec declares the same governed scope and does not declare an ActivationBoundary or ExceptionSpec.

Steps 0–3: Entry and preparation. The entry and preparation stages complete without exclusion or precondition failure. The baseline does not reference excluded dimensions, the `IntentSpec` binding is valid, and `GovernedScope` is present. Both surrounding policies share the baseline resource target `admin_portal` and remain in the filtered working set because neither references an excluded dimension.

Step 4: Analytical Structure Construction. For the `p1` derivation, the relevant applicability facts reduce to two governed cells.

The condition space within `GovernedScope` is partitioned by policy boundaries. Two cells are relevant to the subsequent family checks. They are labelled here as Cell A and Cell B for clarity.

Cell A is (`admin_regular`, `corp_net`, `admin_portal`, `windows`, `browser`, `modern`).

Cell B is (`admin_regular`, `external`, `admin_portal`, `windows`, `browser`, `modern`).

At Cell A, the baseline applies because the cell falls within the baseline's inclusion and outside its exclusion.

At Cell B, the baseline does not apply, because the cell falls within both the baseline's inclusion and its exclusion. Both surrounding policies apply, because the cell falls within their inclusion regions and neither policy declares an exclusion.

Step 5: Family-Specific Candidate Generation. At Cell B, the baseline is absent and both surrounding policies apply. The surrounding grant requirements reduce to strengths 2 and 1, so the effective surrounding strength is 2. The baseline requirement reduces to strength 3. The manual derivation therefore yields one retained `ParallelRoute` candidate for Cell B.

No `ExceptionEscape` expected record follows from this fixture, because no governed cell is left without an applicable policy after exclusions. `StateSpaceGap` is not attempted because the bound `IntentSpec` does not declare an `ActivationBoundary`.

Steps 6–7: Discharge and expected record. No discharge condition applies to the `ParallelRoute` candidate because both sides are reducible within the implemented authentication strength model. The derivation therefore yields one retained `ParallelRoute` candidate at Cell B, with `ExceptionEscape` attempted with no qualifying cells and `StateSpaceGap` not attempted due to absent `ActivationBoundary`.

Correspondence with the expected fixture record. This matches the expected `p1` record in Table 6: retained `ParallelRoute` candidate.

7.2.4 Execution evidence across scenario groups

Table 7 reports the observed record or status for each scenario within the comparison scope defined in Section 7.2.1.

Table 7: Execution evidence across scenario groups

Scenario	Observed record or status	Evidence observed	Evaluation contribution
Canonical scenarios			
`p1`	Retained `ParallelRoute` candidate	External witness cell observed with surrounding strength below the baseline strength	Shows observed retention of a weaker surrounding route within the governed scope
`p1d`	Discharged `ParallelRoute` candidate	Candidate context reached; output records discharge rather than retention	Shows that unsupported comparison contexts are not reported as retained candidates
`p2`	Retained `ExceptionEscape` candidate	The witness cell is included before exclusions by multiple policies and left without an applicable policy after exclusions	Shows exclusion driven loss of applicability across policy composition
`p3`	`StateSpaceGap` structural observation	Uncovered cell inside `ActivationBoundary`; summarized support from multiple mismatch dimensions	Shows structural gap reporting through the implemented `CellBoundaryTrace` surface
`scope`	Scope exclusion path marker	Baseline references an excluded dimension at semantic entry	Shows that abstraction boundary exclusion is recorded before family evaluation
`filter`	Relevant policy filtering path marker	A surrounding policy referencing an excluded dimension is removed from the analyzable set	Shows that filtering changes the working policy set before family checks
Challenge scenarios			
`c1`	Retained `ParallelRoute` candidate	The retained candidate remains observable in the presence of additional surrounding policy noise	Shows that unrelated policy noise does not suppress the intended candidate

Continued on next page

Scenario	Observed record or status	Evidence observed	Evaluation contribution
c2	Retained ExceptionEscape candidate	The witness cell is left uncovered through a distributed exclusion structure	Shows that ExceptionEscape is not limited to a single local exclusion pattern
c3	Discharged ParallelRoute candidate	The candidate context reaches comparison, but unsupported grant controls prevent a retained result	Shows explicit discharge rather than forced classification
c4	StateSpaceGap not attempted path marker	IntentSpec declares GovernedScope but no ActivationBoundary	Shows that StateSpaceGap is gated by its declared precondition
c5	Precondition failure path marker	Required intent input is missing or empty before family evaluation can proceed	Shows preparation failure as a separate record type
Negative guard scenarios			
n1	ParallelRoute attempted with no qualifying cells	The family evaluator is entered, but no cell satisfies the retained or discharged candidate conditions	Shows non-productive family termination without a false candidate
n2	ExceptionEscape attempted with no qualifying cells	Evaluator entered; no qualifying ExceptionEscape record observed	Shows guard behavior for ExceptionEscape
n3	StateSpaceGap attempted with no qualifying cells	Evaluator entered; no StateSpaceGap observation observed	Shows guard behavior for StateSpaceGap
Multi family scenario			
m1	Retained ExceptionEscape candidate and StateSpaceGap structural observation	One witness cell corresponds to both retained ExceptionEscape and StateSpaceGap records; ParallelRoute remains non-productive	Shows preservation of more than one semantic outcome family for the same baseline

The table should be read together with Table 6. Table 6 states the decision rule basis used to derive the expected fixture records, while Table 7 reports the records and evidence observed during execution. The observed records were consistent with the expected fixture records for all fifteen scenarios. This supports fixture level consistency and path observability for the selected scenario set.

Together, the observed records exercise all target outcome categories represented in the prototype comparison surface: semantic scope exclusion, relevant policy filtering, precondition failure, retained and discharged ParallelRoute, retained ExceptionEscape, StateSpaceGap

structural observation, `StateSpaceGap` not attempted, attempted with no qualifying cells outcomes, and one controlled multi-family case. This is a coverage claim about the selected fixture outcomes, not exhaustive branch coverage of the implementation or exhaustive combination coverage across all mechanism families.

7.3 Sensitivity and Limitation Analysis

This section records how two bounded implementation choices affect interpretation of the execution evidence.

7.3.1 Authentication strength only comparison in P1

The `ParallelRoute` evidence is limited by the prototype's authentication strength comparison model. Scenarios `p1d` and `c3` show that unsupported grant controls reach the discharge path, but they do not show how those cases would be classified under a richer comparison model. The evaluation therefore supports the implemented reducibility boundary, not general comparison across all Conditional Access grant control types.

7.3.2 Cell generation from explicit values

The execution evidence depends on the finite cell domain induced by explicit values in the scenario artifacts. The explored condition space is therefore limited to values materialized in the supplied inputs.

The evaluation cannot support claims about condition values that are absent from the fixtures. A broader tenant analysis would require domain expansion or tenant state reconstruction beyond the current prototype boundary.

7.4 Evaluation Validity and Limitations

This section summarizes the main validity limitations that affect interpretation of the evaluation results.

7.4.1 Internal validity

The main internal validity limitation is the lack of an independent oracle. The scenario fixtures, expected output files, and prototype were authored as parts of the same research artifact. The expected outputs therefore encode the thesis interpretation of the architecture rather than an external ground truth. Consistency between observed and expected fixture records shows that the prototype produces the records expected for the controlled scenarios, but it cannot detect a modeling error that is shared by the implementation and the expected output files. The evaluation therefore does not provide an independent validation of the full architecture.

The oracle limitation is reduced, but not eliminated, by making the expected records inspectable in Table 6. The evaluation does not treat expected outputs as opaque values copied from prototype execution. Instead, Table 6 relates each expected fixture record to the decision rule basis used to derive it. This makes the oracle logic explicit and reviewable, but it does not make the oracle independent.

A secondary internal validity limitation concerns selection in fixture design. The scenarios were constructed to exercise specific architectural outcome categories and screening paths, rather than sampled from arbitrary Conditional Access policy configurations. This is appropriate for path coverage, because the evaluation aims to show whether selected architectural outcomes are observable in executable scenarios. However, it also means that the fixture set is not representative of operational policy populations, and the reported consistency results inherit this selection.

7.4.2 Construct validity

The evaluated construct is fixture record consistency for selected semantic outcome paths, not full correctness of Conditional Access governance. The scenario set checks whether the prototype preserves the architectural outcome categories represented in the comparison surface. It does not establish that the taxonomy is complete or that every possible semantic governance failure can be detected.

The comparison surface is also narrower than the full architectural record model. The construct being evaluated is therefore the preservation of selected observable records, not full equivalence between every architectural data structure and its runtime representation.

7.4.3 External validity

Because the evaluation uses synthetic scenario packages rather than live or anonymized Microsoft Entra ID tenant data, the results generalize only to the modeled condition space and the controlled policy compositions represented in the fixtures. They do not show how the prototype behaves under larger tenant policy sets, richer group structures, more diverse condition values, or operational configuration history.

The evaluation also does not include a direct comparison with existing Conditional Access assessment or configuration analysis tools, such as Microsoft's What-If evaluator and community assessment tools discussed in Chapter 3. This is a limitation of the evaluation design. The purpose of the prototype is to examine whether the proposed semantic interaction taxonomy can be operationalized over controlled fixture inputs, not to benchmark detection coverage or operational performance against existing tools. Those tools may provide policy simulation, configuration review, drift assessment, or compliance checks, whereas this thesis evaluates a narrower construct: whether cross-policy semantic interaction can be represented as fixture-level records within a declared governance scope. A structured comparison with existing tools

on shared scenario inputs is therefore left as future work.

7.4.4 Conclusion validity

The conclusion supported by the evaluation is limited to fixture level consistency, represented outcome preservation, and path observability across the fifteen controlled scenarios. The results do not provide statistical evidence about detection accuracy. The negative guard scenarios show that selected family guards suppress findings in controlled cases where no retained candidate or structural observation is expected, but they do not estimate false positive or false negative rates.

The evaluation also does not support claims about scalability. No stress test, large synthetic benchmark, mutation testing campaign, or live tenant execution is included. The observed consistency therefore supports the claim that the implemented prototype follows the expected decision paths for the selected scenarios. It does not support claims about performance, completeness, or platform fidelity beyond the abstraction and fixture set used in this thesis.

Chapter Summary

This chapter evaluated the prototype through the controlled scenario set and interpreted the observed fixture records within the prototype comparison surface. The evaluation supports the bounded claim that the prototype preserves the selected architectural outcome distinctions on the designed fixture set. The claim remains limited by the absence of an independent oracle, the synthetic fixture design, and the restricted scope of the prototype model.

8 Conclusion and Future Work

This thesis examined semantic baseline evaluation in Microsoft Entra ID Conditional Access. The starting point was that identity configuration is not only a collection of isolated policy objects. The effective governance posture of a tenant depends on how baseline policies, surrounding Conditional Access policies, exclusions, and declared intent interact over time.

The thesis proposed a bounded semantic evaluation approach for this problem. The approach separates direct baseline artifact comparison from semantic evaluation of the surrounding policy structure. Within that scope, the work defined a model for representing baseline intent and surrounding policy state, identified selected policy interaction mechanisms, and implemented these ideas in an executable prototype.

The evaluation showed that the prototype can preserve the selected outcome distinctions across the controlled scenario set. The results support the thesis claim that semantic screening can expose policy interaction candidates and structural coverage observations that would not be explained by baseline artifact comparison alone. The claim remains bounded to the modeled scenario state and the selected comparison surface.

8.1 Answer to the Research Question

The research question can be answered as follows. A semantic engine can derive structured records by evaluating a baseline policy artifact together with its declared intent and the surrounding Conditional Access policy structure. The engine must operate within an explicit abstraction boundary, so that only represented policy dimensions and resolved tenant facts are used as the basis for derivation.

Within this bounded model, the engine can distinguish three kinds of semantic outcomes. It can identify weaker alternative routes where surrounding policies provide access under lower assurance than the baseline intent requires. It can identify collective exclusion structures where policies appear to cover a scope before exclusions but leave a subset uncovered after exclusions are applied. It can also identify structural coverage gaps where the declared activation area contains reachable combinations that are not covered by any relevant policy.

The answer is not a single compliance verdict. The semantic engine must preserve different output categories, including retained candidates, discharged candidates, structural observations, and non-productive execution outcomes. This distinction is important because some results indicate a candidate governance concern, while others indicate that evaluation was attempted but did not produce a qualifying finding under the model.

The prototype demonstrated this answer over controlled scenario packages. It showed that the proposed semantic screening architecture can be realized as an executable prototype

that preserves the distinct outcome categories defined by the screening procedure within the selected scenario set. The observed agreement with expected outputs supports fixture level consistency and path observability, but it does not prove completeness of the taxonomy or correctness for live Microsoft Entra ID tenants.

8.2 Limitations

The first limitation is that the prototype is not connected to a live Microsoft Entra ID tenant. It evaluates structured scenario packages rather than tenant snapshots collected from production environments. The results therefore do not show how the approach behaves under the noise and operational complexity of a real tenant.

The second limitation concerns validation. The expected outputs are part of the same research artifact as the scenarios and the prototype. Agreement between expected and observed outputs shows consistency with the modeled interpretation, not independent semantic correctness. This threat is analyzed in Section 7.4.1.

The third limitation is the bounded model. The prototype evaluates only the dimensions admitted by the abstraction boundary defined in Table 2. Signals generated outside that boundary are not represented. The results should therefore not be interpreted as full platform fidelity.

The fourth limitation is that the syntactic comparison engine is not implemented. The thesis architecture distinguishes baseline artifact comparison from semantic evaluation, but the executable prototype evaluates only the semantic part.

8.3 Future Work

One direction for future work is integration with real Microsoft Entra ID tenant snapshots. This would require collecting Conditional Access policy objects, resolving tenant state needed by the admitted dimensions, and translating that state into the representation expected by the semantic engine. Such integration would allow the approach to be evaluated against realistic configuration complexity while still preserving read only analysis.

Another direction is stronger evaluation. The controlled scenario set could be expanded with more diverse policy combinations and independently reviewed expected outputs. Future work should also compare the proposed semantic evaluation approach with existing Conditional Access assessment and configuration analysis tools on a shared set of controlled scenarios. Where possible, this comparison could be extended to anonymized tenant exports. Such a study would help clarify which findings are specific to semantic baseline intent evaluation and which are already covered by existing assessment mechanisms.

A further direction is broadening the semantic model. Future work could examine how to incorporate features outside the present abstraction, including the excluded signal dimensions

defined in Table 2, session controls, and other Conditional Access features. This should not be treated as a simple extension of fields. Each added feature requires a clear account of how it is represented, resolved, and compared.

Additional future work concerns governance support for intent specification. The prototype assumes that the intent artifact is supplied explicitly and correctly. Future work could define templates, validation rules, or authoring workflows for creating and reviewing such intent specifications. This would reduce the risk that semantic evaluation depends on ambiguous or inconsistent intent declarations.

Finally, the approach could be explored beyond Conditional Access baselines. Related identity governance areas such as RBAC role assignments, Privileged Identity Management, Entitlement Management, and cross tenant access may contain similar semantic interaction problems.

8.4 Final Remarks

This thesis does not claim to solve identity governance drift in its full operational form. Its contribution is narrower. It shows how semantic evaluation of Conditional Access baselines can be structured around declared intent, surrounding policy composition, and bounded outcome records. Within that scope, the work defines a model and prototype for deriving selected policy interaction candidates and structural observations within an explicit abstraction boundary.

AI Assistance Disclosure

AI tools, including Claude [2] and ChatGPT [20], were used as supporting tools during the preparation of this thesis. Their use was limited to language refinement, clarity improvement, consistency checks, and editorial feedback.

The academic substance of the thesis was produced by the author. This includes the state-of-the-art review, background material, problem formulation, research design, system architecture, implementation, analysis, and conclusions.

The literature review process, source selection, critical analysis of prior work, and citation selection were carried out by the author. AI tools were not used as a source of generated references.

All AI-assisted outputs were reviewed and verified before inclusion. The author assumes full responsibility for the integrity, accuracy, originality, and academic validity of this work.

References

[1] Vikas Agarwal et al. “Compliance-as-Code for Cybersecurity Automation in Hybrid Cloud”. In: *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*. 2022, pp. 427–437. DOI: 10.1109/CLOUD55607.2022.00066.

[2] Anthropic. *Claude*. https://claude.ai. Accessed: 2026. 2026.

[3] John Backes et al. “Semantic-based automated reasoning for AWS access policies using SMT”. In: *2018 Formal Methods in Computer Aided Design (FMCAD)*. IEEE. 2018, pp. 1–9.

[4] Lee A Barnett et al. “Modeling the AWS authorization engine”. In: *2025 Formal Methods in Computer-Aided Design (FMCAD)*. IEEE. 2025, pp. 1–10.

[5] Andrea Bisegna et al. “Towards Continuous Risk Assessment and Conformance Checking of IdM Deployments”. In: *2025 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE. 2025, pp. 555–560.

[6] Glenn Bruns and Michael Huth. “Access-control policies via Belnap logic: Effective and efficient composition and analysis”. In: *2008 21st IEEE Computer Security Foundations Symposium*. IEEE. 2008, pp. 163–176.

[7] Cloud Native Computing Foundation. *Open Policy Agent*. https://www.openpolicyagent.org. Accessed: 2026.

[8] Thijs van Ede et al. “Detecting anomalous misconfigurations in AWS identity and access management policies”. In: *Proceedings of the 2022 on Cloud Computing Security Workshop*. 2022, pp. 63–74.

[9] Nikhil Ghadge. “Enhancing Identity Management: Best Practices for Governance and Administration”. In: *Available at SSRN 4877656* (2024).

[10] Henry Haverinen et al. “Automating cybersecurity compliance in DevSecOps with open information model for security as code”. In: *Proceedings of the 4th Eclipse Security, AI, Architecture and Modelling Conference on Data Space*. 2024, pp. 93–102.

[11] Alan R Hevner et al. “Design science in information systems research1”. In: *MIS quarterly* 28.1 (2004), pp. 75–106.

[12] Sayed Mohammad Jalali. *Semantic Conditional Access Evaluation Prototype*. 2026. URL: https://github.com/jalali110/EntraID_Semantic_Evaluation.

[13] Matija Kuprešanin and Pavle Subotić. “Ambit: Verification of Azure RBAC”. In: *Proceedings of the 2023 on Cloud Computing Security Workshop*. 2023, pp. 31–40.

[14] Maester365. *Maester*. Open source framework for automated validation of Microsoft security configurations. Accessed: 2026-04-01. URL: https://github.com/maester365/maester.

[15] Microsoft. *Build Conditional Access policies in Microsoft Entra ID*. https://learn.microsoft.com/en-us/entra/identity/conditional-access/concept-conditional-access-policies. Accessed: 2026.

[16] Microsoft. *Conditional Access: What If tool*. Last updated: 2025-11-06. Accessed: 2026-04-01. URL: https://learn.microsoft.com/en-us/entra/identity/conditional-access/what-if-tool.

[17] Microsoft. *Microsoft Entra Conditional Access Optimization Agent*. Last updated: 2026-03-24. Accessed: 2026-04-08. URL: https://learn.microsoft.com/en-us/entra/security-copilot/conditional-access-agent-optimization.

[18] National Institute of Standards and Technology. *OSCAL: Open Security Controls Assessment Language*. https://pages.nist.gov/OSCAL. Accessed: 2026.

[19] NCC Group. *PMapper: Principal Mapper for AWS IAM*. GitHub repository. Accessed: 2026-04-01. URL: https://github.com/nccgroup/pmapper.

[20] OpenAI. *ChatGPT*. https://chatgpt.com. Accessed: 2026. 2026.

[21] Ilia Shevrin and Oded Margalit. "Detecting {Multi-Step}{IAM} attacks in {AWS} environments via model checking". In: *32nd USENIX Security Symposium (USENIX Security 23)*. 2023, pp. 6025–6042.

[22] Abhishek Pratap Singh, Ievgeniia Kuzminykh, and Bogdan Ghita. "Industry perception of security challenges with identity access management solutions". In: *2024 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*. IEEE. 2024, pp. 312–315.

[23] Vlad Stirbu et al. "Toward multiconcern software development with everything as code". In: *IEEE Software* 39.4 (2022), pp. 27–33.

[24] Kennedy A Torkura et al. "Continuous auditing and threat detection in multi-cloud infrastructure". In: *Computers & Security* 102 (2021), p. 102124.

[25] Takumi Yanagawa et al. "A secure framework for continuous compliance across heterogeneous policy validation points". In: *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*. IEEE. 2024, pp. 176–182.

[26] Zhenning Yang et al. "Automated Cloud Infrastructure-as-Code Reconciliation with AI Agents". In: *arXiv preprint arXiv:2510.20211* (2025).

List of Figures

1 Compositional structure of identity enforcement in Microsoft Entra ID 12
2 High level architecture of baseline evaluation 24
3 Conceptual relation between the baseline policy artifact and its associated `IntentSpec` 28
4 Overview of the semantic screening procedure and its principal outcome paths 33
5 Full derivation output structure for one evaluated baseline 40

List of Tables

1 Glossary of terms and abbreviations 4
2 Included and excluded dimensions in the semantic abstraction 29
3 Derived entry fields and roles by mechanism family 41
4 Illustrative derived entry for a retained `ParallelRoute` candidate 44
5 Scenario setup cues used by the prototype 57
6 Manual decision rule basis for expected fixture records 60
7 Execution evidence across scenario groups 63