VisualOFGA: Visualizing Open Fine Grained Authorization Policies

Gaurav Pokharel† Tajkia Nuri Ananna† Amir Masoumzadeh

Albany Lab for Privacy and Security (ALPS), Department of Computer Science, University at Albany, SUNY

† These authors contributed equally to this work

Introduction

OpenFGA enables fine-grained, relationship-based access control through declarative DSL or JSON schemas. Since its release, OpenFGA has rapidly gained adoption and continues to evolve with improved features for developers. However, despite its growing popularity, the system remains largely inaccessible to non-developer users due to the complexity of its DSL and JSON-based schemas.

Current tooling offers only static previews and query-based visualizations, which do little to support broader schema understanding or editing. This creates a usability gap—especially for non-technical stakeholders involved in policy design, review, or auditing.

To bridge this gap, we present **VisualOFGA**—an interactive visualization tool that transforms OpenFGA schemas into modifiable relationship graphs. Our goal is to enhance schema interpretability, support visual debugging, and enable round-trip editing between visual and code-based representations.

Background

Relationship-Based Access Control (ReBAC)

ReBAC is a modern access control model where permissions are derived from the **relationships between entities** (such as users, groups, and resources).

How it works:

- Users gain access based on chains of relationships
- Access logic includes: group memberships, ownerships, delegation
- Enables flexible, real-world sharing (e.g., GitHub, Google Drive)

How it's different:

- RBAC (Role-based, static assignments)
- ABAC (Attribute-based, condition-driven)
- ReBAC (Graph-based, dynamic relationships)

OpenFGA: Scalable ReBAC Engine

OpenFGA is an open-source authorization engine that implements **ReBAC** at scale. Inspired by Google Zanzibar, it allows developers to define and evaluate relationship-based policies.

Core Features:

- Declarative policy definition using **DSL** or **JSON**
- Entity types, relations, and permissions
- Logical composition: `or`, `and`
- Delegation via the `from` keyword
- Tuple-based access evaluation

Example: "User A can write to Repo R if they are a member of a group that owns R."
OpenFGA resolves this via relationship graph traversal at runtime.

Example Use Case: GitHub Repository Access

Modern platforms like **GitHub** illustrate the power of relationship-based access control.
Scenario: A user needs write access to a private repository.

- The repository is owned by an `organization`.
- The user is part of a `team` within that organization.
- The team has been granted `writer` permissions on the repository.

ReBAC Access Chain: `User` → `Team` → `Organization` → `Repo (write)`
OpenFGA captures this logic cleanly using relationship tuples and schema rules—allowing you to visualize and simulate the access path without manually evaluating nested group structures.

Problem Context and VisualOFGA's Contribution

OpenFGA enables scalable ReBAC policies, but understanding its schema structure remains a barrier. We identify several challenges and propose a visual-first solution to improve clarity and control.

⚠ Problem: Schema Complexity

Issue: OpenFGA schemas written in DSL/JSON become hard to interpret, especially with nested relationships and delegation.

✓ Solution: Visual Schema Rendering — we convert schemas into interactive graphs to make relationships and logic more transparent.

⚠ Problem: Limited Visualization

Issue: OpenFGA currently offers only static type previews or query-based path graphs, which don't help understand full schema logic.

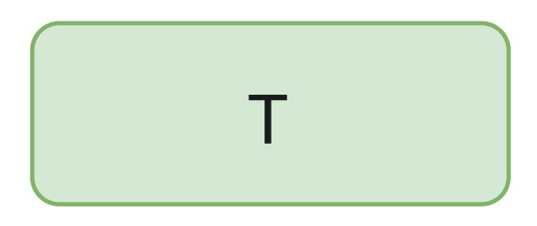
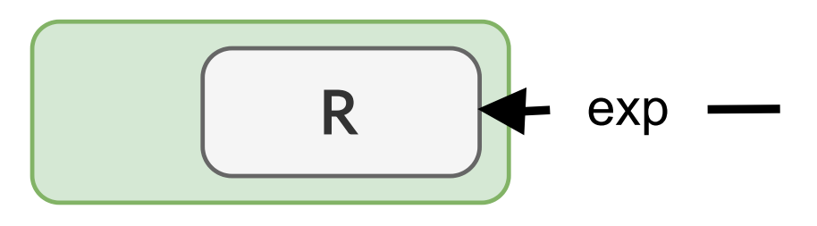
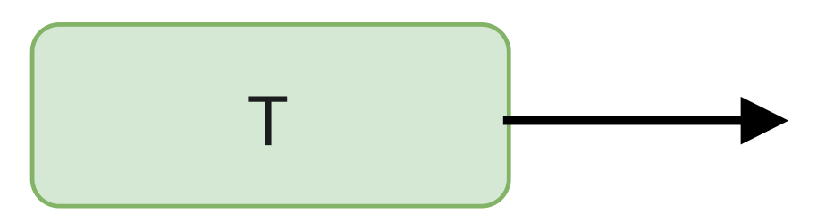
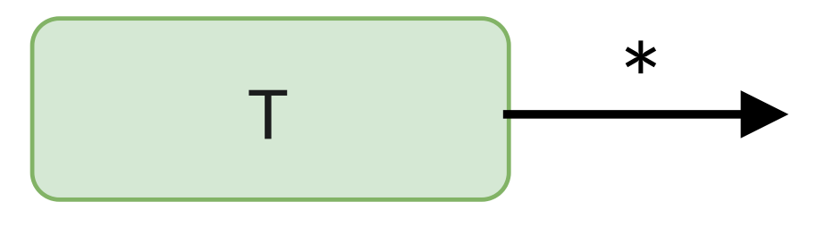
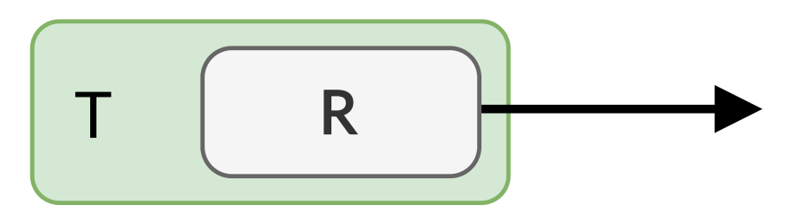
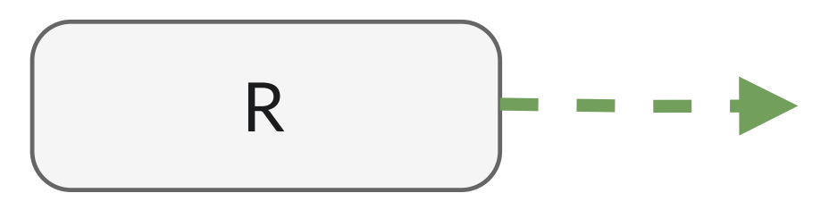
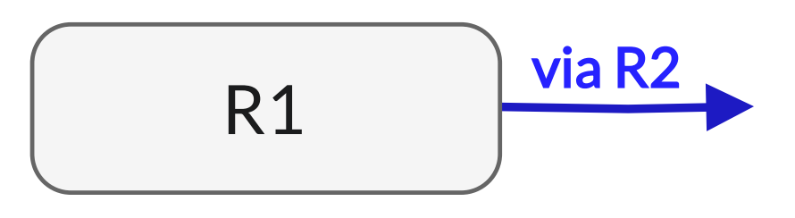
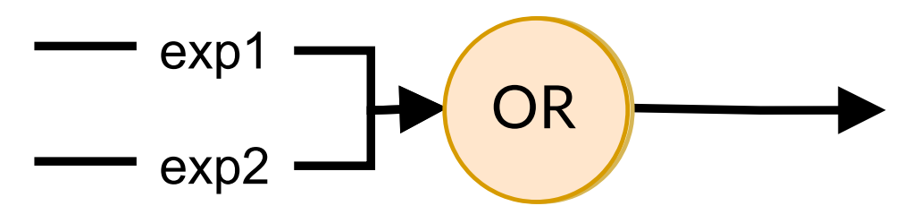
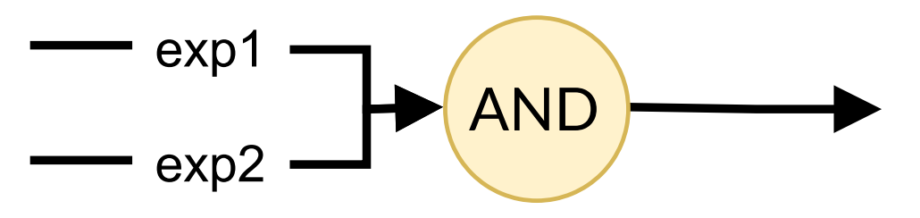
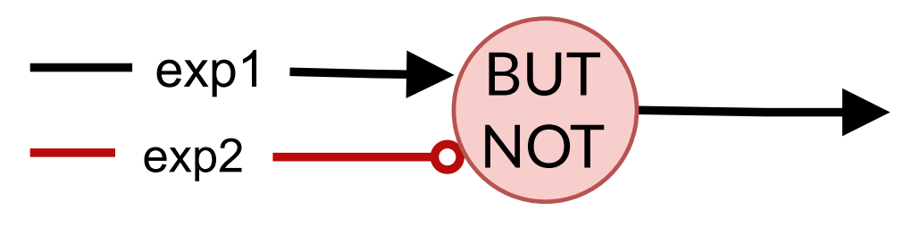
✓ Solution: Full Relationship Flow Visualization — we visualize all entity types, relations, inheritance, and delegation paths.

⚠ Problem: No Visual Editing

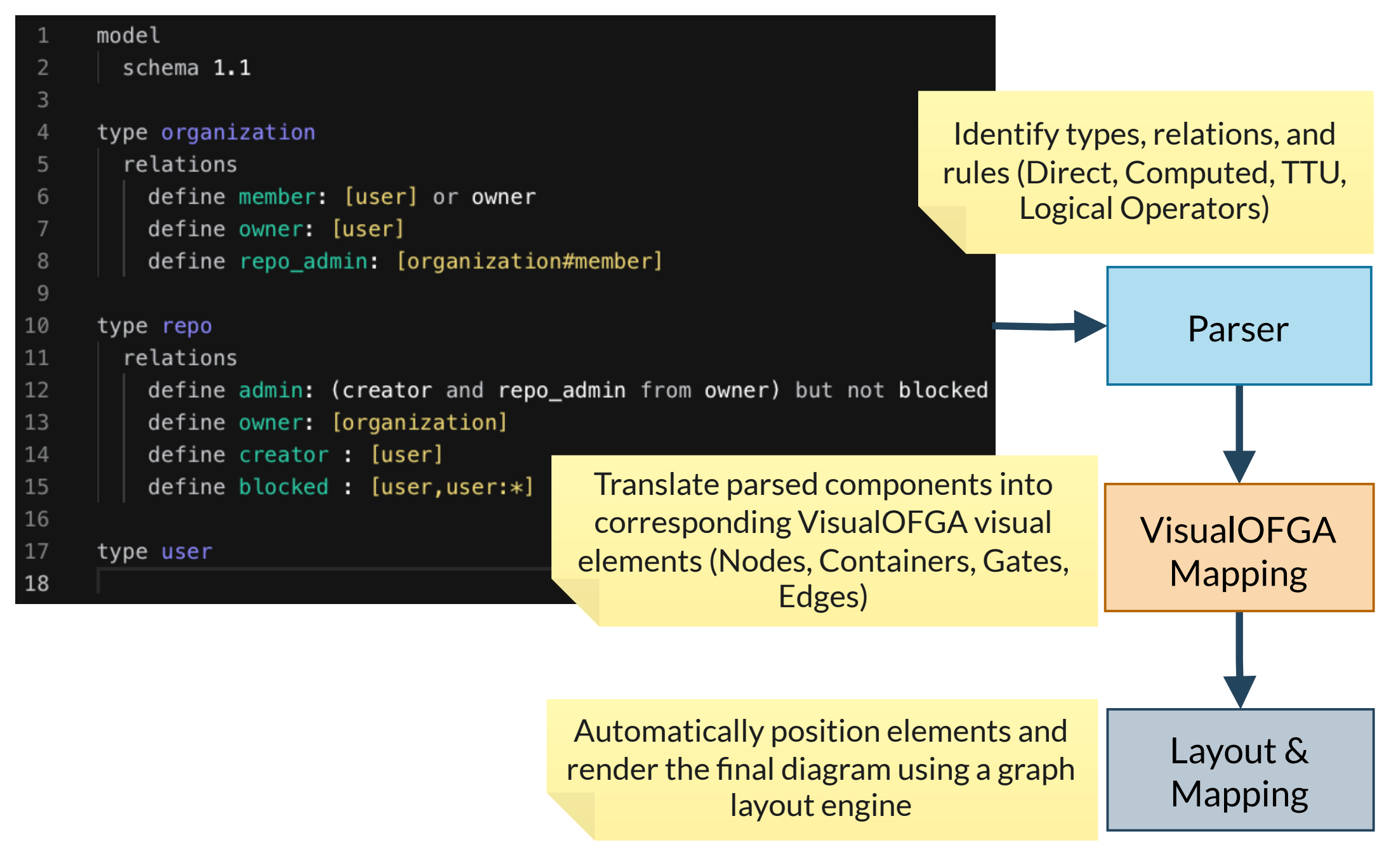
Issue: There's no way to modify policies visually — editing must be done manually in DSL or JSON.

✓ Solution: Graph-to-Schema Reversibility — users can modify the visual graph and convert it back into valid schema definitions.

Visual Notation Summary

Category	OpenFGA Construct / Terminology	Formal SDL Syntax	Visualization
Definition	Type Definition	type T	
	Relation Definition	define R: exp	
Base Expression	Direct Relationship Type Restrictions (Direct Type Assignment)	[T]	
	Direct Relationship Type Restrictions (Type Bound Public Access)	[T:*]	
	Direct Relationship Type Restrictions (Direct Userset Assignment)	[T#R]	
	Referencing Other Relations On The Same Object (Computed Userset)	R	
	Referencing Relations On Related Objects (TupleToUserset)	R1 from R2	
Logical Operator	Union Operator (or)	exp1 or exp2	
	Intersection Operator (and)	exp1 and exp2	
	Exclusion Operator (but not)	exp1 but not exp2	

Methodology

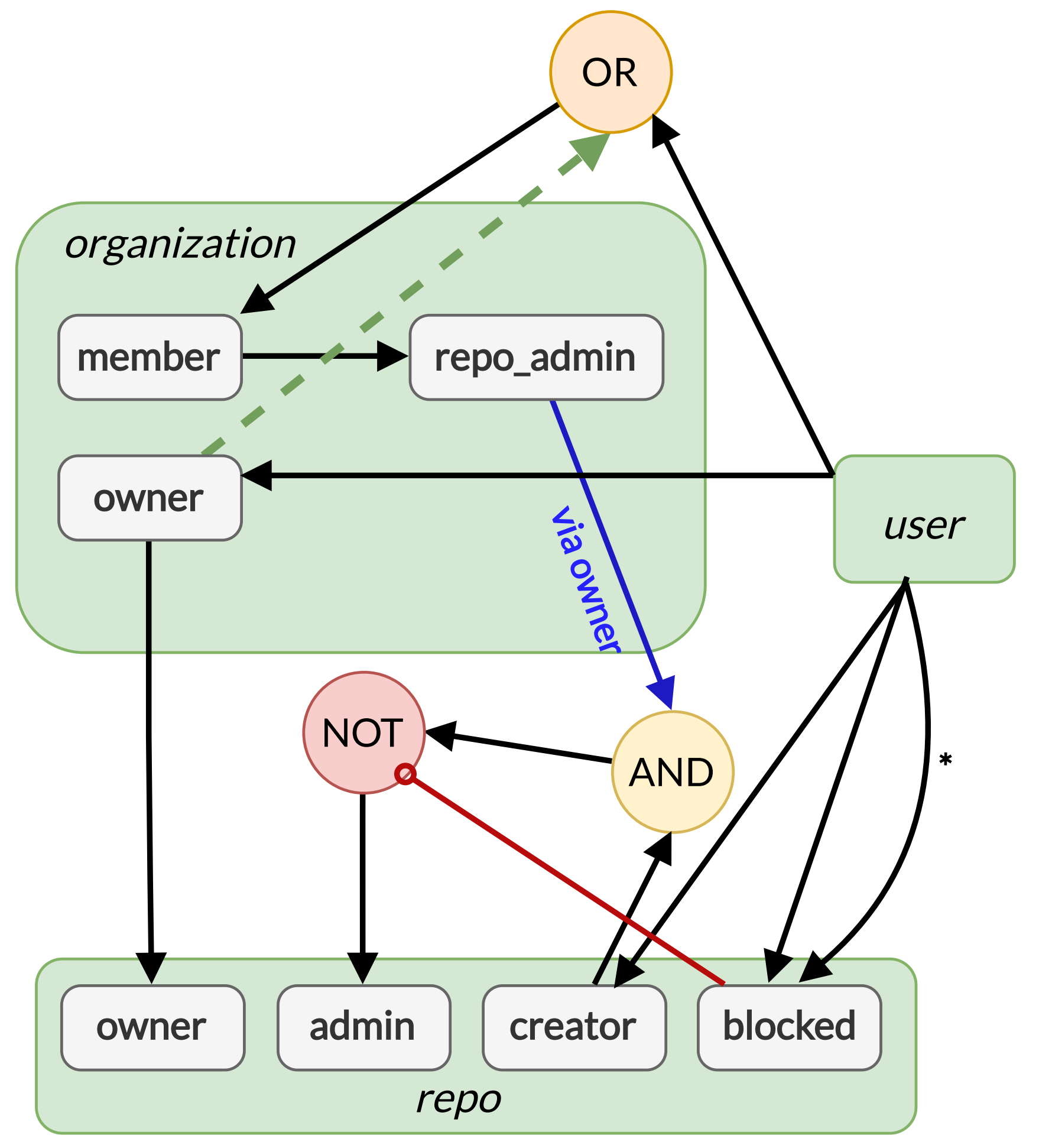

Identify types, relations, and rules (Direct, Computed, TTU, Logical Operators)
Parser
Translate parsed components into corresponding VisualOFGA visual elements (Nodes, Containers, Gates, Edges)
VisualOFGA Mapping
Automatically position elements and render the final diagram using a graph layout engine
Layout & Mapping

```
model
  schema 1.1

type organization
  relations
    define member: [user] or owner
    define owner: [user]
    define repo_admin: [organization#member]

type repo
  relations
    define admin: (creator and repo_admin from owner) but not blocked
    define owner: [organization]
    define creator : [user]
    define blocked : [user,user:*]

type user
```



Future Work

The VisualOFGA notation provides a robust foundation for enhanced OpenFGA schema comprehension and tooling. Future research directions include:

- **Schema and Tuple Graph Integration:** Develop an interactive visualization that integrates the schema graph with a graph representing stored relationship tuples. This would allow users to explore the interplay between the model's definition and its instance data.
- **Query Dependency Paths:** Enable display of dependency paths for authorization queries, showing how tuples and schema elements determine outcomes.
- **Schema Evolution and Impact Analysis:** Introduce "what-if" analysis capabilities, enabling users to visualize the potential impact of proposed schema modifications. This could involve visual diffing between schema versions and simulating query outcomes under different model iterations.

References

[1] Philip WL Fong. Relationship-based access control: protection model and policy language. In *Proceedings of the first ACM conference on Data and application security and privacy*, pages 191–202, 2011.

[2] Ruoming Pang, Ramon Caceres, Mike Burrows, Zhifeng Chen, Pratik Dave, Nathan Germer, Alexander Golynski, Kevin Graney, Nina Kang, Lea Kissner, et al. Zanzibar:{Google's} consistent, global authorization system. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 33–46, 2019.

[3] The Linux Foundation contributors. Openfga. `https://openfga.dev/`. [Online; accessed 15-April-2025].