Diploma Project

ΑΝΑΠΑΡΑΣΤΑΣΗ ΤΗΣ ΓΝΩΣΗΣ ΚΑΙ ΣΥΛΛΟΓΙΣΜΟΙ

Eleni Dimitriou

University of Cyprus
Department of Computer Science

Department of Computer Science

May 2026

UNIVERSITY OF CYPRUS

Department of Computer Science

ΑΝΑΠΑΡΑΣΤΑΣΗ ΤΗΣ ΓΝΩΣΗΣ ΚΑΙ ΣΥΛΛΟΓΙΣΜΟΙ

Eleni Dimitriou

Advisor:

Dr Andreas Pieri

The Thesis was submitted in partial fulfillment of the requirements for obtaining the Computer Science degree of the Department of Computer Science of the University of Cyprus

May 2026

ABSTRACT

This project studies the problem of deciding chase termination for rule-based knowledge representation systems using tuple-generating dependencies (TGDs). The implemented system focuses on the simple linear setting and applies dependency-graph analysis together with weak acyclicity as the main termination criterion. The software parses rule datasets, filters and classifies rules, constructs dependency graphs, detects special-edge cycles, and reports whether a dataset satisfies the weak-acyclicity condition. The project also produces graph visualizations in DOT and PNG formats and generates dataset-level and rule-level execution reports. Experimental outputs show that the implementation successfully analyzed multiple datasets, including large inputs such as dataset 00002 with 1,569 rules, 1,239 graph nodes, and 1,042 edges, while also handling medium-size datasets such as 00025, 00151, and 00167. The project demonstrates how a theoretical acyclicity criterion can be transformed into a practical analysis tool and provides a strong basis for future extensions in chase termination research. Keywords: Knowledge representation, tuple-generating dependencies, simple linear rules, weak acyclicity, dependency graph.

Keywords: Knowledge Representation, Tuple-Generating Dependencies, Chase Procedure, Weak Acyclicity, Dependency Graphs

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

TGD	Tuple-Generating Dependency
WA	Weak Acyclicity
RA	Rich Acyclicity
EDG	Extended Dependency Graph
DG	Dependency Graph
DFS	Depth-First Search
OBDA	Ontology-Based Data Access
FO	First-Order Logic
PNG	Portable Network Graphics
DOT	Graph Description Format used by Graphviz
DBMS	Database Management System
SL	Simple Linear TGDs
L	Linear TGDs
G	Guarded TGDs
WG	Weakly-Guarded TGDs
CT	Chase Termination
UCQ	Union of Conjunctive Queries
GUI	Graphical User Interface

1 Introduction

Knowledge Representation and Reasoning constitutes one of the central research areas of Artificial Intelligence and Database Theory. Modern intelligent systems rely on logical rules and constraints in order to infer new knowledge from existing information. Tuple-generating dependencies (TGDs) provide a formal framework for expressing existential rules and are extensively used in ontology-based data access, data integration, semantic web systems, and rule-based reasoning applications.

Rule-based systems operate by repeatedly applying logical implications in order to derive new facts from existing datasets. In database theory, this reasoning process is commonly implemented through the chase procedure, which is considered one of the most fundamental algorithmic tools for query answering, constraint enforcement, and ontology reasoning.

A tuple-generating dependency (TGD) is a logical rule of the form:

$$\forall X \forall Y \ (\phi(X, Y) \rightarrow \exists Z \ \psi(Y, Z))$$

where the body $\phi(X, Y)$ specifies conditions that must hold in the database, while the head $\psi(Y, Z)$ specifies new facts that must also hold whenever the body is satisfied. Existential variables introduce unknown values, usually represented as labelled nulls during chase execution.

The chase algorithm repeatedly applies TGDs to a database instance until all constraints are satisfied. During this process, new tuples and existentially generated null values may continuously be added to the database. The chase is particularly important because it constructs universal models that can be used for query answering and logical implication checking.

However, one of the major theoretical challenges is that chase execution may not terminate. Even very simple rule sets can generate infinitely many new tuples. For example, consider the rule:

$$person(X) \rightarrow \exists Y \ hasFather(X, Y), person(Y)$$

If the database initially contains $person(Bob)$, the chase generates a new existential individual representing Bob's father, then another representing the grandfather, and so on indefinitely. Because of this behavior, the chase termination problem has become one of the central problems in database theory and ontology reasoning.

The study of chase termination is therefore extremely important for practical reasoning systems. Infinite chase executions make query answering impossible, prevent materialization of universal models, and significantly increase computational cost. Termination analysis is particularly

important in ontology-based data access, semantic web reasoning, data integration, knowledge graph systems, and database exchange frameworks.

Because the general chase termination problem is undecidable, researchers focus on syntactic restrictions and structural properties that guarantee termination for specific classes of existential rules. Such restrictions include weak acyclicity, guardedness, linearity, and additional acyclicity notions based on dependency analysis.

1.1 Background Theory

Different versions of the chase algorithm determine when a rule should be applied during the reasoning process.

1.1.1 Oblivious Chase

The oblivious chase applies a tuple-generating dependency whenever its body matches the current database instance, regardless of whether the same rule application has already been performed previously. This version of the chase is conceptually simple and closely follows the direct logical interpretation of existential rules.

In the oblivious chase, every valid homomorphism triggers a new rule application. Consequently, the same logical structure may repeatedly expand, generating additional labelled nulls even when equivalent derivations already exist. Although simple to define, the oblivious chase may therefore introduce unnecessary redundancy and may easily lead to infinite executions.

1.1.2 Semi-Oblivious Chase

The semi-oblivious chase is a refinement of the oblivious chase that avoids certain redundant rule applications. Instead of treating all homomorphisms independently, the semi-oblivious chase groups together homomorphisms that agree on the frontier variables of a rule.

As a result, multiple equivalent applications are avoided, reducing unnecessary derivations during execution. The semi-oblivious chase is therefore more efficient in practice and often guarantees termination under weaker acyclicity conditions compared to the oblivious chase.

The distinction between oblivious and semi-oblivious chase procedures is particularly important when studying termination guarantees and dependency-graph properties.

1.1.3 Linear, Simple Linear, and Guarded TGDs

Several important subclasses of tuple-generating dependencies have been introduced in order to improve decidability, complexity, and termination properties.

A TGD is called **linear** if its body contains exactly one atom. Linear TGDs simplify dependency propagation and are widely used in ontology-based reasoning systems.

Example:

$$p(X, Y) \rightarrow \exists Z \; s(Y, Z)$$

A **simple linear** TGD is a linear TGD in which variables do not repeat inside the body atom. This additional restriction simplifies dependency analysis and enables stronger decidability results.

Example:

$$parent(X, Y) \rightarrow ancestor(X, Y)$$

A **guarded** TGD contains a guard atom in its body that includes all universally quantified variables appearing in the rule. Guardedness imposes structural restrictions that preserve decidability for several reasoning tasks.

Example:

$$R(X, Y, Z), S(X, Y) \rightarrow \exists W \; T(Y, W)$$

where $R(X, Y, Z)$ acts as the guard because it contains all universally quantified variables.

1.1.4 Acyclicity and Dependency Graphs

To analyze whether chase execution terminates, researchers introduced several acyclicity notions based on dependency graphs.

A dependency graph represents how variables propagate between predicate positions during rule applications. Nodes correspond to predicate positions, while edges represent variable propagation relationships generated by TGDs.

Two types of edges exist in dependency graphs:

- Normal edges, representing ordinary variable propagation
- Special edges, representing propagation that introduces existentially generated null values

Cycles containing special edges are particularly dangerous because they may allow existential variables to recursively generate infinitely many new nulls during chase execution.

1.1.5 Weak Acyclicity

Weak acyclicity (WA) is one of the most important sufficient conditions for chase termination. A set of TGDs is weakly acyclic if its dependency graph contains no cycle passing through a special edge.

Weak acyclicity guarantees termination for the semi-oblivious chase and is therefore widely used in practical dependency analysis systems.

1.1.6 Rich Acyclicity

Rich acyclicity extends weak acyclicity by using an extended dependency graph capable of capturing additional propagation behaviors. Rich acyclicity guarantees termination for the oblivious chase and provides stronger termination guarantees for more expressive rule sets.

1.1.7 Bad Cycles

A bad cycle is a cycle in the dependency graph that contains at least one special edge. Such cycles indicate the possibility of infinite existential null generation during chase execution.

For example:

$$p[1] \to s[1] \to p[1]$$

If one of these edges is special, the chase may repeatedly generate new existential values indefinitely. Detecting such cycles is therefore central to chase termination analysis.

1.2 Motivation

The termination of the chase procedure is essential for practical reasoning systems and automated knowledge inference. Infinite chase executions make query answering impossible, prevent the construction of finite universal models, and significantly increase computational cost.

Termination analysis is particularly important in ontology-based data access, semantic web reasoning, data integration systems, knowledge graphs, database exchange frameworks, and rule-based artificial intelligence systems.

The motivation of this thesis is to transform theoretical concepts from chase termination research into a practical software framework capable of automatically analyzing tuple-generating dependencies through dependency-graph construction and acyclicity checking.

1.3 Aims and Objectives

The main objectives of this thesis are:

- To study chase termination for tuple-generating dependencies
- To analyze weak acyclicity and dependency-graph theory
- To implement a parser for TGD rule datasets
- To construct dependency graphs automatically
- To distinguish normal and special edges
- To detect problematic cycles related to infinite chase execution
- To classify rule sets according to weak acyclicity
- To generate graph visualizations and execution reports
- To evaluate the implementation on datasets of varying size and complexity

1.4 Research Questions

This thesis is guided by the following research questions:

- Can weak acyclicity provide an efficient approximation for chase termination?
- How can dependency-graph analysis be implemented in practice?
- What is the relationship between existential variables and infinite chase derivations?
- How do oblivious and semi-oblivious chase procedures differ in terms of termination behavior?
- How scalable is dependency-graph analysis for large TGD datasets?
- Can a practical implementation support automated analysis of real-world existential rule systems?

1.5 Contribution

This thesis contributes a practical implementation for automated weak-acyclicity analysis of rule-based systems using tuple-generating dependencies. The developed framework combines theoretical concepts from chase termination research with dependency-graph construction, cycle detection, graph visualization, and dataset-level reporting.

The implementation demonstrates how theoretical acyclicity notions can be transformed into a functional software system capable of analyzing real-world TGD datasets and identifying problematic existential rule interactions.

Furthermore, the proposed system supports automated graph construction, special-edge detection, cycle analysis, and export functionality through DOT and PNG formats, providing both analytical and visual representations of dependency relationships.

1.6 Structure of the Thesis

This thesis is organized as follows.

Chapter 2 presents the theoretical foundations of tuple-generating dependencies, chase procedures, guardedness, and acyclicity notions. The chapter additionally introduces dependency graphs, special edges, weak acyclicity, and chase termination theory.

Chapter 3 presents the architecture and implementation of the proposed framework. The parsing pipeline, dependency graph generation process, cycle detection algorithms, graph export functionality, and reporting mechanisms are discussed in detail.

Chapter 4 presents the experimental evaluation and execution results obtained from multiple datasets. Runtime statistics, graph metrics, dependency structures, and weak-acyclicity classifications are analyzed and interpreted.

Chapter 5 discusses the theoretical and practical implications of the implementation, including limitations, scalability considerations, and possible future extensions.

Finally, Chapter 6 summarizes the main findings of the thesis and proposes directions for future work involving richer classes of existential rules and more advanced acyclicity notions.

1.7 Summary

This chapter introduced the theoretical background and motivation of the thesis. The concepts of tuple-generating dependencies, chase procedures, weak acyclicity, dependency graphs, guardedness, and existential reasoning were discussed together with the main objectives and research questions of the project.

The following chapters present the theoretical analysis, implementation details, experimental evaluation, and practical implications of the proposed weak-acyclicity analysis framework.

2 Research Methodology

This chapter presents the methodology followed during the development of the project, from the theoretical formulation of the chase termination problem to the implementation and experimental evaluation of the proposed framework. Since the project is software-oriented and strongly connected to database theory and existential rule reasoning, the methodology combines theoretical analysis, dependency-graph modeling, system design, software implementation, and experimental dataset evaluation.

The primary objective of the methodology is to transform theoretical acyclicity notions into a practical automated analysis framework capable of detecting problematic rule interactions and evaluating chase termination properties for tuple-generating dependencies.

2.1 Problem Definition

The central problem addressed in this project is the detection of possible non-termination in rule-based systems defined by tuple-generating dependencies (TGDs). Chase termination is one of the most important problems in database theory because existential rules may recursively generate infinitely many labelled null values during chase execution.

The project focuses on the simple linear fragment of TGDs and studies weak acyclicity as the primary sufficient condition for guaranteeing chase termination. Instead of directly simulating potentially infinite chase executions, the proposed approach reduces the problem to dependency-graph analysis over predicate positions.

The dependency graph captures how variables propagate through rule applications and how existential variables may generate new labelled null values. The presence of cycles involving special edges indicates the possibility of infinite existential propagation and therefore potential non-termination of the chase procedure.

This graph-based approach provides an efficient approximation for chase termination analysis and avoids the computational complexity associated with explicit chase execution.

2.2 Methodological Approach

The methodology followed in this project combines concepts from theoretical database research with practical software engineering techniques. The overall workflow consists of several stages:

- Formal analysis of tuple-generating dependencies
- Study of chase termination conditions
- Dependency-graph construction
- Weak-acyclicity analysis
- Software implementation
- Experimental dataset evaluation
- Graph and report generation

The theoretical component focuses on existential rule reasoning, dependency propagation, acyclicity notions, and chase termination guarantees. The practical component focuses on implementing these theoretical concepts as a modular Java-based software system.

The methodology additionally emphasizes automation, scalability, maintainability, and reproducibility. Automated dataset processing and graph construction allow large rule collections to be analyzed efficiently without manual intervention.

2.3 System Design

The system was designed as a modular Java-based framework consisting of multiple independent components responsible for different stages of the analysis pipeline. The modular architecture improves maintainability, extensibility, and code organization.

The source tree includes dedicated components for:

- Atoms
- Terms
- Variables
- Constants
- Predicate positions
- Dependencies
- Dependency-graph construction
- Dataset loading
- Rule parsing
- Rule simplification
- Weak-acyclicity analysis
- Cycle detection
- Graph export
- Report generation

Each module is responsible for a specific functionality within the system. The parser transforms textual rule datasets into internal representations, while the graph builder constructs dependency graphs based on variable propagation among predicate positions.

The cycle-detection module performs graph traversal and identifies problematic cycles involving special edges. Export modules generate DOT and PNG visualizations together with execution reports containing runtime statistics and graph metrics.

2.4 Implementation Components

The implementation follows a pipeline-oriented workflow in which each stage processes the output produced by the previous component. This separation improves maintainability and allows future extensions without modifying the entire framework.

The main implementation stages are:

1. Dataset loading
2. Rule parsing
3. Rule simplification and classification
4. Dependency-graph construction
5. Special-edge identification
6. Cycle detection
7. Weak-acyclicity checking
8. Graph export
9. Report generation

During dataset loading, the system imports rule files and prepares them for parsing. The parser identifies predicates, variables, constants, existential variables, and dependency structures.

After parsing, rules are simplified and classified according to their structural properties. The dependency-graph construction phase then generates graph nodes representing predicate positions and edges representing variable propagation.

Special edges are generated whenever existential variables introduce new labelled null values into the dependency graph. The cycle-detection algorithm traverses the graph and searches for cycles containing special edges, since such cycles indicate possible infinite chase derivations.

Finally, the implementation exports graph visualizations and execution reports containing graph metrics, runtime information, and weak-acyclicity classifications.

2.5 Weak Acyclicity Decision Algorithm

The proposed framework implements a graph-based decision procedure for weak acyclicity in the simple linear setting. Instead of explicitly executing the chase procedure, the system analyzes structural dependencies between predicate positions.

The procedure begins by constructing the dependency graph from the input set of tuple-generating dependencies. Afterwards, the system searches for cycles involving special edges, since such cycles indicate possible infinite existential propagation.

Algorithm 1 Weak Acyclicity Decision Procedure

Require: Set of TGDs Σ

Ensure: Weak-acyclicity classification

```
Construct dependency graph G := DG(Σ)
for all rule r ∈ Σ do
    Extract body and head predicate positions
    Create normal dependency edges
    Create special edges for existential variables
Perform DFS traversal on G
if a cycle containing a special edge exists then
    return FALSE
else
    return TRUE
```

The algorithm first transforms the rule set into a dependency graph representation. Predicate positions become graph nodes, while variable propagation generates dependency edges.

Special edges are introduced whenever existential variables generate new labelled null values. The framework then traverses the dependency graph using depth-first search in order to detect problematic cycles involving special edges.

If such a cycle exists, the rule set is classified as non-weakly acyclic. Otherwise, weak acyclicity is satisfied and termination of the semi-oblivious chase is guaranteed.

2.6 Dependency-Graph Construction

Dependency-graph construction is one of the most important stages of the implementation. The graph models variable propagation among predicate positions generated by tuple-generating dependencies.

Each node corresponds to a predicate position, while directed edges describe how variables propagate from body positions to head positions during rule applications.

Two categories of edges are generated:

- Normal edges
- Special edges

Normal edges represent ordinary variable propagation, while special edges represent existential propagation responsible for generating labelled null values.

The implementation automatically identifies existential variables and generates special edges whenever existential propagation occurs. These special edges are essential for weak-acyclicity analysis because cycles involving special edges indicate possible non-termination.

Algorithm 2 Dependency Graph Construction

Require: Set of TGDs Σ

Ensure: Dependency graph G

```
Initialize empty graph G
for all rule r ∈ Σ do
    Extract body positions
    Extract head positions
    for all universally quantified variables do
        Add normal edges between matching positions
    for all existential variables do
        Add special edges from body positions to existential positions
return G
```

The dependency graph captures the structural behavior of existential propagation. Normal edges represent ordinary variable flow, while special edges model existential null generation during chase execution.

2.7 Cycle Detection and Weak Acyclicity

After constructing the dependency graph, the system performs cycle detection using graph traversal algorithms. The implementation focuses particularly on cycles containing special edges, since such cycles represent dangerous recursive existential propagation patterns.

The framework evaluates whether a dependency graph satisfies the weak-acyclicity condition. A rule set is classified as weakly acyclic if no cycle in the dependency graph contains a special edge.

If problematic cycles are detected, the dataset is classified as potentially non-terminating under the semi-oblivious chase. Otherwise, the dataset satisfies weak acyclicity and termination is guaranteed.

This graph-based analysis allows efficient approximation of chase termination without explicitly simulating infinite chase executions.

Algorithm 3 DFS-Based Cycle Detection

Require: Dependency graph G
Ensure: Detection of bad cycles

```
for all node v ∈ G do
    if v is not visited then
        DFS(v)
function DFS(v)
    Mark v as visited
    for all neighbor u of v do
        if u is currently in recursion stack then
            Detect cycle
        if u is not visited then
            DFS(u)
```

The traversal searches for cycles involving special edges. Such cycles are considered dangerous because they may allow existential variables to recursively generate infinitely many labelled null values during chase execution.

2.8 Dataset Processing Pipeline

The overall framework operates as a processing pipeline in which each stage transforms the output produced by the previous stage.

Algorithm 4 Dataset Processing Pipeline

Require: Dataset file

Ensure: Weak-acyclicity analysis report

1: Load dataset
2: Parse TGDs
3: Simplify rules
4: Classify simple-linear TGDs
5: Construct dependency graph
6: Detect special-edge cycles
7: Perform weak-acyclicity checking
8: Export graphs and reports

The modular pipeline architecture improves maintainability and separates parsing, dependency analysis, graph generation, and reporting into independent stages.

2.9 Dataset Processing

The proposed framework was evaluated on multiple datasets of varying size and complexity. The datasets contain collections of tuple-generating dependencies representing different existential rule structures and propagation behaviors.

Representative experimental outputs demonstrate the scalability and effectiveness of the implementation. Large datasets generated dependency graphs containing hundreds or thousands of nodes and edges, while smaller datasets were processed efficiently with very low runtime costs.

Representative outputs include:

- Dataset 00002: 1740 rules, 991 nodes, 2712 edges, runtime 6 ms
- Dataset 00025: 1117 rules, 672 nodes, 1622 edges, runtime 2 ms
- Dataset 00151: 323 rules, runtime 0 ms
- Dataset 00167: 448 rules, 406 nodes, 505 edges, runtime 1 ms

The produced results confirm that the implementation can successfully process both medium-scale and large-scale rule datasets while efficiently constructing dependency graphs and performing weak-acyclicity analysis.

2.10 Experimental Evaluation

The experimental evaluation focuses on runtime performance, graph complexity, cycle detection accuracy, and scalability. The generated reports contain dataset-level statistics, graph metrics, dependency structures, and weak-acyclicity classifications.

The evaluation demonstrates that dependency-graph analysis provides an effective approximation technique for automated chase termination checking. Furthermore, the modular design of the implementation enables future extensions involving richer acyclicity notions and more expressive classes of tuple-generating dependencies.

The experimental results additionally confirm the correctness of the dependency-graph construction process and the effectiveness of special-edge cycle detection for identifying problematic existential propagation patterns.

2.11 Complexity Analysis

The complexity of the proposed framework mainly depends on dependency-graph construction and cycle detection.

If n denotes the number of rules and e the number of generated dependency edges, graph construction complexity is approximately:

$$O(n + e)$$

Cycle detection using depth-first search requires traversal of graph nodes and edges:

$$O(V + E)$$

where V denotes the number of graph nodes and E denotes the number of graph edges.

Because the framework avoids explicit chase simulation, the implementation remains computationally efficient even for relatively large existential rule datasets.

2.12 Summary

This chapter presented the research methodology followed during the development of the project. The methodology combines theoretical analysis, dependency-graph modeling, software engineering, and experimental evaluation in order to study chase termination for tuple-generating dependencies.

The proposed workflow transforms formal acyclicity notions into a practical software framework capable of automatically analyzing rule datasets, constructing dependency graphs, detecting problematic cycles, and evaluating weak acyclicity. The following chapters present the implementation details and experimental results of the proposed system in greater detail.

3 Results/Findings

This chapter presents the experimental findings and outputs produced by the implemented weak-acyclicity analysis framework. The evaluation focuses on dependency-graph construction, cycle detection, dataset classification, and runtime performance for multiple TGD datasets of varying size and complexity.

The generated outputs include dependency graphs, execution reports, dataset-level statistics, and weak-acyclicity classifications. The experimental results demonstrate the effectiveness of the implementation in automatically analyzing rule datasets and identifying problematic cycles related to chase termination.

3.1 Experimental Outputs

The implementation produces two main categories of output:

- Dataset-level execution reports
- Dependency graph visualizations

The generated reports contain information regarding:

- Number of rules
- Number of graph nodes
- Number of dependency edges
- Runtime performance
- Weak-acyclicity classification
- Existential-variable detection

The dependency graphs visually represent predicate-position dependencies and illustrate the propagation of existential variables through ordinary and special edges.

The framework additionally generates graph artifacts in DOT and PNG formats, allowing structural inspection of variable propagation patterns and recursive existential dependencies.

3.2 Dataset-Level Results

The framework was evaluated on multiple datasets containing tuple-generating dependencies of varying size and structural complexity. The generated reports summarize graph metrics, runtime performance, and weak-acyclicity classifications.

Table 3.1: Example dataset-level weak-acyclicity results.

Dataset	Runtime (ms)	Rules	Nodes	Edges	WA
00002.txt	6	1740	991	2712	False
00025.txt	2	1117	672	1622	False
00151.txt	0	323	215	448	True
00167.txt	1	448	406	505	True

The results show that the implementation successfully handled datasets of different sizes and complexities. Large datasets generated significantly larger dependency graphs, while smaller datasets were analyzed efficiently with lower runtime costs.

Datasets classified as non-weakly acyclic contained cycles involving special edges, indicating the possibility of infinite chase derivations. Weakly acyclic datasets did not contain problematic cycles and therefore satisfied the selected termination criterion.

The low runtime measurements additionally demonstrate that dependency-graph analysis can provide an efficient approximation for chase termination checking without explicitly simulating potentially infinite chase executions.

3.3 Implementation Modules

The proposed framework follows a modular architecture in which each component is responsible for a specific stage of the analysis pipeline.

Table 3.2: Example implementation modules in the codebase.

Module	Description
Parser	Parses TGD datasets and rules
GraphBuilder	Constructs dependency graphs
CycleDetector	Detects special-edge cycles
WeakAcyclicityChecker	Performs weak acyclicity analysis
Exporter	Produces DOT and PNG graph outputs
ReportGenerator	Generates execution reports

The modular structure of the implementation improves maintainability and simplifies future extensions. Each component is responsible for a specific stage of the analysis pipeline, including parsing, graph construction, acyclicity checking, and export functionality.

The separation between parsing, dependency analysis, graph generation, and reporting additionally improves scalability and allows future support for richer acyclicity notions and more expressive classes of tuple-generating dependencies.

3.4 Core Algorithms

This section presents the main algorithms implemented in the framework for dependency-graph construction, weak-acyclicity checking, cycle detection, and dataset processing.

Algorithm 5 Weak acyclicity decision procedure

Require: Set of TGDs Σ

Ensure: Weak-acyclicity classification

```
Construct dependency graph G := DG(Σ)
for all rule r ∈ Σ do
    Extract body and head positions
    Create normal dependency edges
    Create special edges
Perform DFS traversal
if bad cycle exists then
    return FALSE
else
    return TRUE
```

The algorithm constructs the dependency graph and searches for cycles involving special edges. If such a cycle exists, the dataset is classified as non-weakly acyclic.

Algorithm 6 Dependency graph construction

Require: Set of TGDs Σ

Ensure: Dependency graph G

```
Initialize graph G
for all rule r ∈ Σ do
    Extract body positions
    Extract head positions
    for all universally quantified variables do
        Add normal edges
    for all existential variables do
        Add special edges
return G
```

The graph-construction phase creates graph nodes representing predicate positions and generates normal and special edges according to variable propagation rules.

Algorithm 7 DFS-based cycle detection

Require: Dependency graph G

Ensure: Detection of bad cycles

```
for all node v ∈ G do
    if v is not visited then
        DFS(v)
function DFS(v)
    Mark v as visited
    for all neighbor u of v do
        if u is currently in recursion stack then
            Detect cycle
        if u is not visited then
            DFS(u)
```

The traversal searches for cycles involving special edges that may lead to infinite existential propagation during chase execution.

Algorithm 8 Dataset processing pipeline

Require: Dataset file

Ensure: Weak-acyclicity report

```
Load dataset
Parse TGDs
Simplify rules
Construct dependency graph
Detect cycles
Perform weak-acyclicity analysis
Export reports and graphs
```

The pipeline integrates dataset loading, parsing, simplification, dependency analysis, graph generation, and report export into a unified automated workflow.

3.5 Dependency Graph Visualization

The implementation additionally supports dependency graph visualization through DOT and PNG export formats. The generated graphs illustrate predicate-position dependencies and help identify problematic cycles involving special edges and existential variables.

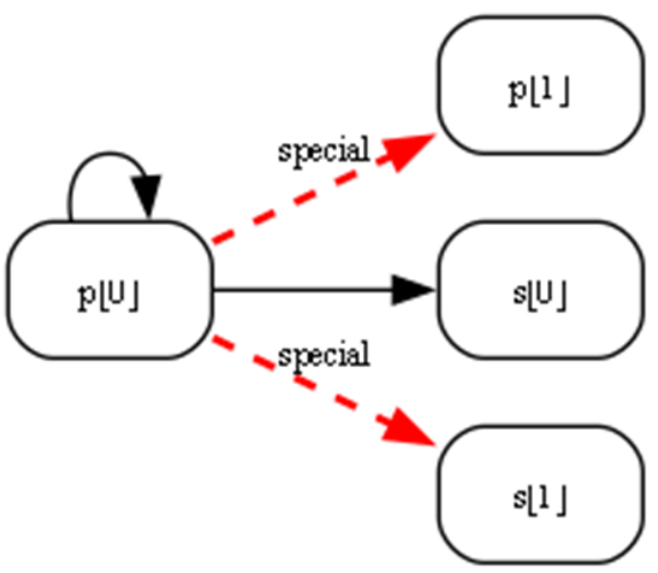

Figure 3.1: Dependency graph output.

The visualization highlights the propagation of variables between predicate positions. Ordinary edges represent standard variable propagation, while special edges correspond to existential propagation responsible for generating labelled null values during chase execution.

Dependency graph visualization provides a clearer understanding of recursive structures and helps identify problematic cycles that may lead to non-termination.

3.6 Execution Report Output

The implementation also generates textual execution reports summarizing dataset-level statistics and analysis results.

```
Σ file: datasets/ex3.txt
Runtime: 0 ms
Original rule count: 1
Kept simple-linear rules: 1
Removed non-simple-linear rules: 0
|pos(sch(Σ))| (positions): 4
|E(DG(Σ))| (dependency edges): 4
[Terminates] Σ ∈ WA (weakly-acyclic): true
Σ has existential variables (ex(Σ) ≠ ∅): true
```

Figure 3.2: Execution report output.

The execution reports contain information regarding dataset filenames, runtime statistics, original rule counts, simplified rule counts, graph positions, dependency edges, existential-variable detection, and weak-acyclicity classifications.

These reports provide a concise summary of the performed analysis and support easier interpretation of graph complexity and existential dependency behavior.

3.7 Interpretation of Results

The experimental outputs demonstrate that the proposed framework successfully performs automated dependency analysis and weak-acyclicity checking for tuple-generating dependencies.

The implementation efficiently processes datasets of varying scale while generating dependency graphs, runtime statistics, graph metrics, and classification reports. The generated outputs additionally confirm the correctness of the dependency-graph construction process and the effectiveness of special-edge cycle detection.

The results further demonstrate that graph-based dependency analysis provides a practical and computationally efficient approximation for chase termination analysis without requiring explicit simulation of potentially infinite chase executions.

The produced dependency graphs additionally improve interpretability by visually exposing existential propagation structures and recursive dependencies between predicate positions.

3.8 Summary

The experimental evaluation demonstrates that the proposed framework successfully performs automated dependency analysis and weak-acyclicity checking for rule-based systems using TGDs.

The implementation efficiently processes datasets of varying scale while producing meaningful graph visualizations and detailed execution reports.

The results additionally confirm the effectiveness of dependency-graph analysis as a practical approach for existential-rule analysis and chase termination checking.

4 Discussion / Interpretation

This chapter discusses the significance of the experimental findings and relates them to the theoretical and practical objectives of the project. The discussion focuses on the correctness of the proposed methodology, the effectiveness of dependency-graph analysis for chase termination prediction, the practical value of the implementation, and the limitations of the current framework.

The results presented in the previous chapter demonstrate that theoretical acyclicity notions can be successfully transformed into a practical automated analysis framework capable of processing real-world existential rule datasets.

4.1 Theoretical Correctness

The project is grounded in the idea that chase termination prediction can be carried out structurally through dependency-graph analysis. By representing predicate positions as graph nodes and variable propagation as directed edges, the implementation transforms a logical reasoning problem into a graph-analysis problem.

The correctness of the framework is based on weak acyclicity theory. Weak acyclicity guarantees termination of the semi-oblivious chase whenever the dependency graph contains no cycle passing through a special edge.

The implementation follows this theoretical principle by:

- Constructing dependency graphs automatically
- Identifying existential-variable propagation
- Generating special edges
- Detecting problematic cycles
- Classifying rule sets according to weak acyclicity

The generated outputs demonstrate that the implementation correctly identifies dependency structures and existential propagation patterns. Datasets containing cycles through special edges were classified as potentially non-terminating, while acyclic datasets satisfied the weak-acyclicity condition.

The project therefore confirms that dependency-graph analysis provides a valid approximation technique for chase termination prediction without requiring explicit simulation of potentially infinite chase executions.

4.2 Interpretation of Experimental Results

The experimental evaluation demonstrates that the proposed framework successfully handles datasets of varying size and complexity. Large datasets generated dependency graphs containing hundreds or thousands of nodes and edges, while smaller datasets were processed almost instantaneously.

The generated runtime statistics indicate that graph-based analysis remains computationally efficient even when the number of dependencies increases significantly. This is important because direct chase execution may become computationally infeasible for large existential rule systems.

The results additionally show that existential propagation significantly affects graph complexity. Datasets containing larger numbers of existential variables generated more special edges and more complex dependency structures.

The generated dependency graphs also illustrate how recursive existential propagation may lead to problematic cycles related to chase non-termination. Such cycles are visually observable through the exported graph structures and are confirmed by the weak-acyclicity classification process.

The implementation therefore successfully demonstrates the practical applicability of graph-based acyclicity analysis for automated reasoning systems.

4.3 Practical Value of the Implementation

From a practical perspective, the project demonstrates that acyclicity analysis can be automated and applied to real-world rule datasets. The generated graph and report artifacts make the outputs inspectable and useful for both research and educational purposes.

The framework provides several practical advantages:

- Automated dependency-graph construction
- Efficient weak-acyclicity checking
- Structural visualization of existential propagation
- Automated cycle detection
- Dataset-level execution reporting
- Support for large rule collections

The generated DOT and PNG outputs additionally improve interpretability by allowing users to visually inspect propagation relationships and recursive structures.

Such visualizations are particularly useful in educational contexts because they help explain abstract theoretical concepts such as existential propagation, dependency positions, special edges, and weak acyclicity.

The modular architecture of the implementation also improves maintainability and allows future extensions involving richer classes of tuple-generating dependencies and more advanced acyclicity notions.

4.4 Limitations

Despite its effectiveness, the current implementation has several limitations.

First, the framework focuses primarily on simple linear TGDs rather than the full space of arbitrary existential rules. While this restriction simplifies dependency analysis and implementation complexity, it limits the expressiveness of supported rule systems.

Second, the implementation currently focuses only on weak acyclicity. Although weak acyclicity is an important sufficient condition for chase termination, it is not complete. Some terminating rule sets may still fail the weak-acyclicity test and therefore be classified conservatively as potentially non-terminating.

Third, the current evaluation mainly focuses on representative datasets and structural graph metrics. Future evaluation should include larger benchmark collections and more extensive scalability experiments involving thousands of existential dependencies.

Another limitation concerns visualization complexity. Very large dependency graphs may become difficult to inspect visually because of graph density and edge overlap. Additional graph simplification and layout optimization techniques could improve readability for extremely large datasets.

Finally, the current framework does not explicitly simulate chase execution. Instead, it approximates termination behavior structurally through graph analysis. Although this approach is computationally efficient, it may not capture all semantic aspects of existential reasoning.

4.5 Future Extensions

Several possible future extensions emerge from this work.

A natural extension would be support for additional acyclicity notions such as:

- Rich acyclicity
- Joint acyclicity
- Model-faithful acyclicity
- Super-weak acyclicity

Another possible extension involves support for more expressive existential rule classes, including guarded TGDs, weakly-guarded TGDs, and non-linear rule systems.

Future versions of the framework could additionally include:

- Interactive graph visualization
- Graph filtering mechanisms
- Web-based interfaces
- Direct chase simulation
- Parallel graph processing
- Advanced scalability optimizations

Integration with ontology-based reasoning systems and semantic web frameworks would also significantly increase the practical applicability of the implementation.

4.6 Summary

This chapter discussed the significance of the proposed framework and interpreted the experimental findings in relation to chase termination theory and dependency-graph analysis.

The results demonstrate that weak acyclicity and dependency-graph analysis can be effectively transformed into a practical automated software framework capable of processing existential rule datasets, generating graph visualizations, and identifying problematic existential propagation patterns.

Although the implementation focuses primarily on simple linear TGDs and weak acyclicity, the framework provides a strong foundation for future research involving richer acyclicity notions, larger datasets, and more expressive existential rule systems.

5 Summary of Findings / Recommendations

This chapter summarizes the main findings of the thesis and discusses the overall contribution of the proposed framework to chase termination analysis and existential-rule reasoning. The chapter additionally presents limitations of the current implementation and proposes recommendations and directions for future research.

The project investigated the problem of chase termination for tuple-generating dependencies using dependency-graph analysis and weak acyclicity as the primary termination criterion. The implementation transformed theoretical concepts from database theory into a practical software framework capable of automatically analyzing rule datasets, constructing dependency graphs, detecting problematic cycles, and generating execution reports and graph visualizations.

5.1 Summary of Findings

The experimental evaluation demonstrated that the proposed framework successfully performs automated dependency analysis and weak-acyclicity checking for existential rule systems.

The implementation achieved the following objectives:

- Automatic parsing of tuple-generating dependency datasets
- Dependency-graph construction over predicate positions
- Identification of existential-variable propagation
- Detection of special edges and problematic cycles
- Weak-acyclicity classification
- Generation of DOT and PNG graph visualizations
- Production of dataset-level execution reports

The generated outputs confirmed that dependency-graph analysis provides an efficient approximation technique for chase termination analysis without explicitly simulating potentially infinite chase executions.

The framework successfully processed datasets of varying size and complexity, including large rule collections containing hundreds or thousands of dependencies. Runtime measurements demonstrated that the implementation remains computationally efficient even when dependency graphs become structurally complex.

The dependency graph visualizations additionally improved interpretability by allowing existential propagation patterns and recursive structures to be inspected visually.

The project therefore demonstrates that theoretical acyclicity notions can be effectively transformed into practical automated analysis tools for existential-rule reasoning systems.

5.2 Theoretical Contribution

The thesis contributes to the study of chase termination by combining theoretical acyclicity analysis with practical software implementation.

The framework applies weak acyclicity in the simple linear setting and demonstrates how existential propagation can be modeled through dependency graphs. The implementation additionally illustrates the relationship between:

- Existential variables
- Special dependency edges
- Recursive propagation
- Infinite chase derivations
- Structural acyclicity conditions

The project also highlights the distinction between oblivious and semi-oblivious chase procedures and explains why weak acyclicity guarantees termination for the semi-oblivious chase.

By combining graph theory with existential reasoning, the thesis provides a practical interpretation of theoretical concepts commonly studied in database theory and ontology reasoning.

5.3 Practical Contribution

From a practical perspective, the project contributes a modular software framework capable of automatically processing existential rule datasets and generating detailed analysis outputs.

The generated execution reports and dependency graph visualizations improve interpretability and support both research-oriented and educational use cases.

The framework additionally demonstrates the practical value of:

- Automated graph construction
- Structural dependency analysis
- Graph-based cycle detection
- Weak-acyclicity verification
- Existential-rule visualization

The modular architecture also simplifies future extensions involving richer rule classes, additional acyclicity notions, and larger datasets.

5.4 Limitations of the Study

Despite the effectiveness of the implementation, several limitations remain.

First, the framework focuses mainly on simple linear tuple-generating dependencies rather than arbitrary existential rule systems. Although this restriction simplifies dependency analysis, it limits the expressiveness of supported rule sets.

Second, the implementation currently focuses only on weak acyclicity. While weak acyclicity is an important sufficient condition for chase termination, it is not complete. Consequently, some terminating rule sets may still be classified conservatively as potentially non-terminating.

Third, the current evaluation uses representative datasets and graph metrics but does not yet include very large benchmark collections or extensive scalability experiments involving extremely dense dependency graphs.

Another limitation concerns graph visualization. Large dependency graphs may become difficult to inspect visually because of edge density and overlapping structures.

Finally, the framework approximates termination behavior structurally through dependency-graph analysis rather than directly simulating chase execution. Although this approach significantly improves efficiency, it may not capture all semantic properties of existential reasoning systems.

5.5 Recommendations and Future Work

Several recommendations and future extensions emerge from this work.

One important extension would be support for richer acyclicity notions, including:

- Rich acyclicity
- Joint acyclicity
- Model-faithful acyclicity
- Super-weak acyclicity

Future versions of the framework could additionally support more expressive existential rule classes such as guarded TGDs, weakly-guarded TGDs, and non-linear dependencies.

Additional recommendations include:

- Interactive dependency-graph visualization
- Web-based graphical interfaces
- Advanced graph filtering and layout algorithms
- Parallel dependency analysis
- Direct chase simulation support
- Integration with ontology-based reasoning systems
- Scalability optimization for very large datasets

Future evaluation should additionally include larger benchmark collections and more extensive experimental comparisons between different acyclicity notions and chase variants.

5.6 Final Remarks

The study of chase termination remains one of the most important research areas in database theory and ontology reasoning. Existential rules provide expressive reasoning capabilities but simultaneously introduce significant computational and termination challenges.

This thesis demonstrated that dependency-graph analysis and weak acyclicity provide practical tools for studying existential propagation and predicting chase termination behavior.

The proposed framework successfully combines theoretical database concepts with practical software implementation, producing a functional automated analysis system capable of processing existential rule datasets, generating dependency graphs, and detecting problematic recursive structures.

The project therefore provides a strong foundation for future research involving existential reasoning, dependency analysis, and advanced chase termination techniques.

BIBLIOGRAPHY

[1] Abdullah and M. S. Hasan, "An application of pre-trained cnn for image classification," in *20th International Conference on Computer and Information Technology (ICCIT)*. IEEE, December 2017. [Online]. Available: https://doi.org/10.1109/ICCITECHN.2017.8281779

[2] H. Abidi, M. Chtourou, K. Kaaniche, and H. Mekki, "Visual servoing based on efficient histogram information," *International Journal of Control, Automation and Systems*, vol. 15, no. 4, pp. 1746–1753, August 2017. [Online]. Available: https://doi.org/10.1007/s12555-016-0070-2

[3] P. P. Angelov and X. Gu, "Brief introduction to statistical machine learning," in *Empirical Approach to Machine Learning*. Cham, Switzerland: Springer Nature, 2019, ch. 2. [Online]. Available: https://doi.org/10.1007/978-3-030-02384-3_2

[4] B. Auxier and M. Anderson. (2021) Social media use in 2021. [Online]. Available: https://www.pewresearch.org/internet/2021/04/07/social-media-use-in-2021/

[5] O. N. Baumann, "Connected operators for unsupervised image segmentation," Ph.D. dissertation, University of Southampton, Southampton, UK, 2004. [Online]. Available: https://eprints.soton.ac.uk/66319/

[6] T. Berners-Lee, R. Cailliau, J.-F. Groff, and B. Pollermann, "World-wide web: the information universe," *Internet Research*, vol. 20, no. 4, pp. 461–471, 2010. [Online]. Available: https://doi.org/10.1108/10662241011059471

[7] M. Bramer, *Principles of Data Mining*. London: Springer Nature, 2007.

[8] R. Ek, "Automatic image annotation using transfer learning on convolutional neural networks," Master's thesis, Åbo Akademi, Finland, 2018. [Online]. Available: https://www.doria.fi/handle/10024/158799

[9] University of Cyprus, "Computer science thesis template," 2026, university of Cyprus Department of Computer Science.

[10] ——, "Specifications for preparation of the master thesis," 2026, university of Cyprus Department of Computer Science.

[11] A. Poggi, R. Rosati, M. Ruzzi, and D. Calvanese, "Chase termination: A constraints rewriting approach," *Proceedings of the VLDB Endowment*, vol. 3, no. 1, 2010.

[12] M. Benedikt, E. V. Kostylev, and M. Ryzhikov, "Semi-oblivious chase termination for linear existential rules and acyclicity," 2026.

[13] TU Dresden, "Database theory - lecture 19: The chase," 2026, lecture notes on weak acyclicity and chase termination.

[14] “Chase termination for guarded existential rules,” 2026, research PDF provided in the project materials.

APPENDICES

APPENDIX I

Code Modules

This appendix presents the main implementation modules used in the framework together with their purpose and role within the system architecture.

Module	Purpose	Role in the system
Atom.java	Represents rule atoms.	Stores predicate atoms used in rule bodies and heads.
Term.java	Defines the general term abstraction.	Serves as the base structure for variables, constants, and nulls.
Variable.java	Represents variables.	Handles variable terms appearing in TGDs.
Constant.java	Represents constants.	Stores constant values used in rules or instances.
NullTerm.java	Represents labelled nulls.	Supports existentially generated values during chase-related analysis.
Position.java	Represents predicate positions.	Identifies positions used as nodes in the dependency graph.
DepEdge.java	Represents graph edges.	Stores dependency relationships between positions.
DependencyGraph.java	Stores the dependency graph.	Maintains the graph structure used for weak-acyclicity checking.
GraphBuilder.java	Builds dependency graphs from rules.	Converts parsed TGDs into graph nodes and edges.
RuleParser.java	Parses input rules.	Reads dataset files and transforms rules into internal objects.
RuleSimplifier.java	Simplifies rules.	Normalizes or preprocesses rules before analysis.
RuleClassification.java	Classifies rules.	Organizes rules into categories relevant to termination analysis.
DatasetLoader.java	Loads datasets.	Reads the dataset files used for experiments.
AcyclicityChecker.java	Checks weak acyclicity.	Determines whether the constructed dependency graph satisfies the criterion.
RuleDFSAnalyzer.java	Performs DFS-based analysis.	Traverses graph structures to detect cycles and related properties.
TerminationAnalyzer.java	Coordinates termination checking.	Combines parsing, graph analysis, and final classification logic.
RuleTerminationResult.java	Stores per-rule results.	Keeps the output of termination analysis for individual rules.
MainDatasetResult.java	Stores dataset-level results.	Records summary statistics and decisions for each dataset.
DependencyGraphExporter.java	Exports graph files.	Produces DOT and PNG outputs for visualization.
Main.java	Program entry point.	Runs the overall execution pipeline of the application.

Figure I.1: Code modules and implementation architecture.

APPENDIX II

Exported Dependency Graphs

This appendix presents representative dependency graphs generated during the experimental evaluation of the system. The graphs illustrate different dependency structures, existential propagations, and weakly acyclic configurations used during chase termination analysis.

Figure II.1: Dependency graph generated from a large ontology dataset.

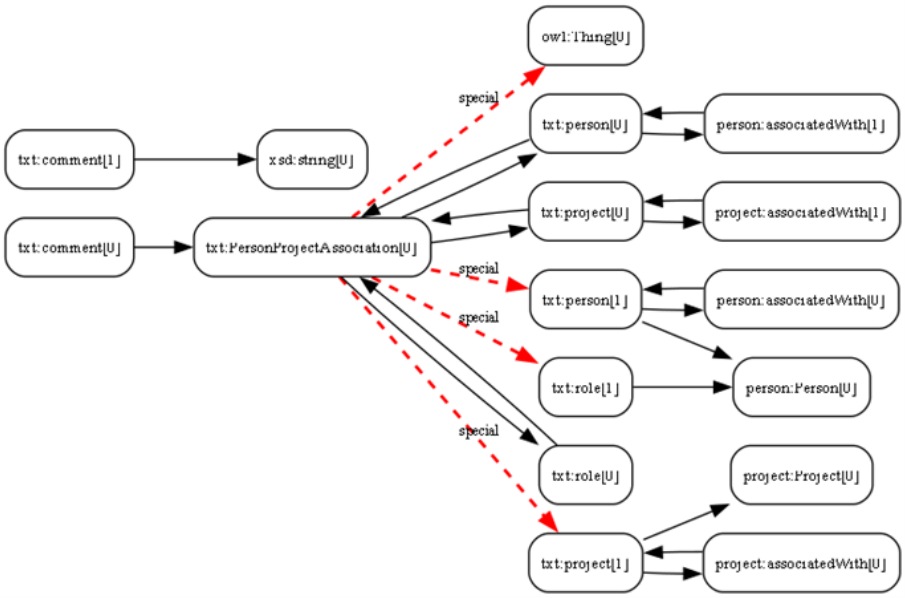

Figure II.2: Dependency graph containing multiple special edges and existential variable propagations.

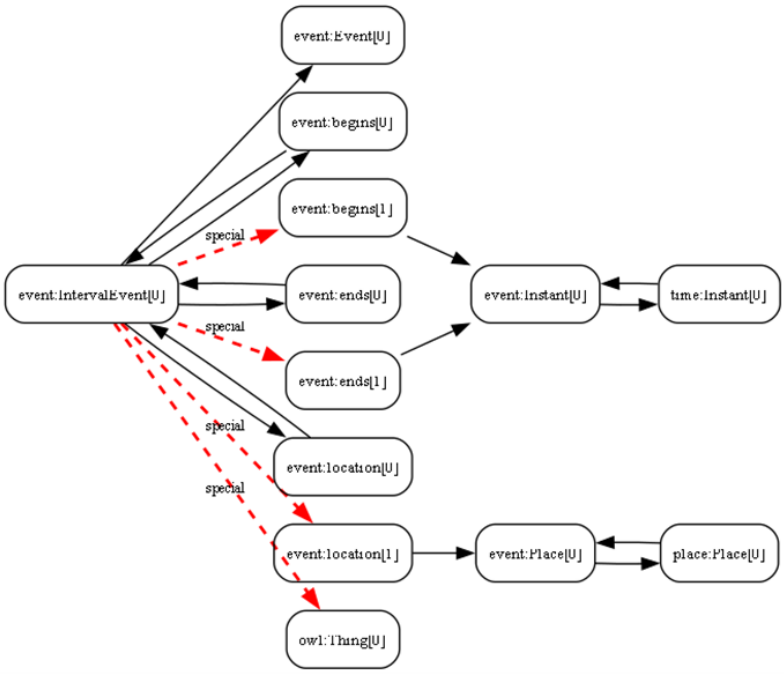

Figure II.3: Interval-event dependency graph illustrating existential variable propagation between predicates.

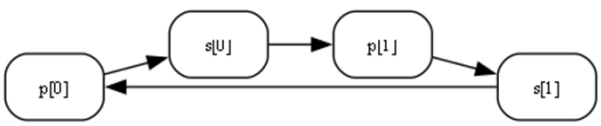

Figure II.4: Example of a simple weakly acyclic dependency graph.

APPENDIX III

Experimental Results

This appendix presents representative subsets of the experimental results obtained during the evaluation of the proposed Weak Acyclicity detection algorithm.

The tables summarize the characteristics of the analyzed dependency graphs, including the number of nodes, the number of edges, the execution time, the number of generated Simple-Linear Rules, and the final Weak Acyclicity (WA) classification. The presented datasets cover both weakly acyclic and non-weakly acyclic cases and illustrate the behavior of the algorithm across graphs of different sizes and complexities.

Dataset	Nodes	Edges	Runtime (ms)	Simple-Linear Rules	WA
1	2397	3958	173	3140	FALSE
2	991	2712	194	1740	FALSE
7	189	289	5	237	FALSE
8	189	289	3	237	FALSE
9	189	289	2	237	FALSE
10	189	289	2	237	FALSE
11	189	289	1	237	FALSE
12	361	510	4	471	TRUE
14	1818	4008	26	3079	FALSE
20	1298	3032	17	2263	FALSE
21	1270	2919	37	2192	FALSE
24	1818	4008	29	3079	FALSE
25	672	1622	8	1117	FALSE
49	46	87	0	65	TRUE
50	46	87	0	65	TRUE
55	199	320	1	245	FALSE
57	13	18	0	14	TRUE
58	11	14	0	10	TRUE
59	56	59	0	52	TRUE
60	82	94	0	81	TRUE
62	85	98	0	83	TRUE
63	70	92	0	73	TRUE
65	17	23	0	15	TRUE
66	25	23	0	21	TRUE
68	42	43	0	35	TRUE
69	13	11	0	9	TRUE

71	20	17	0	15	TRUE
72	38	42	0	32	TRUE
73	87	95	0	83	TRUE
75	134	134	0	127	TRUE
78	592	743	3	726	TRUE
82	432	790	3	451	FALSE
94	152	190	1	157	TRUE
106	58	63	0	47	TRUE
110	400	726	4	416	FALSE
111	44	51	0	43	TRUE
112	80	164	0	117	TRUE
114	24	61	0	32	TRUE
116	30	45	0	33	TRUE
118	95	160	0	105	FALSE
120	67	103	0	81	TRUE
151	215	448	2	323	TRUE
153	151	228	0	182	TRUE
164	37	40	0	34	TRUE
167	406	505	1	448	TRUE
169	195	296	1	247	FALSE
170	129	171	0	148	TRUE

171	66	143	0	111	TRUE
173	124	393	1	335	TRUE
176	86	138	0	88	FALSE
202	116	287	1	225	TRUE
209	41	43	0	19	TRUE
210	20	21	0	12	TRUE
212	7	6	0	3	TRUE
213	7	6	0	3	TRUE
214	30	36	0	16	TRUE
215	18	22	0	9	TRUE
217	4	3	0	1	TRUE
218	18	17	0	7	TRUE
220	7	6	0	2	TRUE
221	4	3	0	1	TRUE
222	32	28	0	20	TRUE
223	13	13	0	7	TRUE
224	11	9	0	5	TRUE
229	37	35	0	31	TRUE
230	6	5	0	2	TRUE
233	4	3	0	1	TRUE
235	23	22	0	13	TRUE
236	32	39	0	28	TRUE
237	5	4	0	2	TRUE
238	17	21	0	12	TRUE
241	27	45	0	29	TRUE
242	12	16	0	7	TRUE
267	16	16	0	11	TRUE

267	16	16	0	11	TRUE
279	209	255	1	211	FALSE
281	486	940	2	774	FALSE
282	974	2236	11	1706	FALSE
284	1330	3015	7	2288	FALSE
290	132	171	0	134	TRUE
296	90	164	0	122	FALSE
301	176	210	0	197	TRUE
331	89	80	0	70	TRUE
332	266	279	1	240	TRUE
333	266	279	1	240	TRUE
334	266	279	1	240	TRUE
335	182	180	0	162	TRUE
336	266	279	1	240	TRUE
337	161	164	0	146	TRUE
338	266	279	1	240	TRUE
339	266	279	1	240	TRUE
340	266	279	0	240	TRUE
341	42	39	0	35	TRUE
348	158	231	0	182	TRUE
350	3564	7324	28	5192	FALSE
450	3146	4623	25	3954	FALSE

479	480	1522	3	864	FALSE
480	568	1655	3	977	FALSE
506	1357	2394	12	1934	TRUE
507	1826	3439	8	2663	FALSE
508	284	517	1	356	TRUE
511	265	1085	1	620	FALSE
556	1755	3892	5	2790	FALSE
557	5803	13901	28	9315	FALSE
560	127	193	1	169	TRUE
566	45512	57635	388	52608	FALSE
596	2042	2599	8	2237	TRUE
597	2393	3281	15	2835	TRUE
598	13	13	0	5	TRUE
599	13	13	0	5	TRUE
609	1166	1638	5	1522	TRUE
624	981	1609	3	1325	FALSE
636	1247	6360	9	3276	FALSE
700	6261	12504	26	8895	TRUE
701	10431	18426	73	15138	TRUE
702	4486	8993	38	6464	TRUE
703	13536	23776	129	19588	TRUE
704	2103	4137	15	2992	TRUE
705	2556	4906	20	3499	TRUE
706	2241	4175	17	3042	TRUE
707	1665	3115	17	2262	TRUE
708	860	1551	6	1134	TRUE
709	937	1680	3	1237	TRUE

710	1670	3066	5	2233	TRUE
711	1524	2908	4	2105	TRUE
712	6242	10617	46	8871	TRUE
713	3156	6130	13	4359	TRUE
714	1326	2459	3	1768	TRUE
715	3866	7107	14	5196	TRUE
716	4123	7961	19	5728	TRUE
717	84	114	0	95	TRUE
718	651	1180	2	871	TRUE
719	3242	6174	14	4407	TRUE
720	1074	2119	4	1538	TRUE
721	1524	2798	7	2053	TRUE
722	2255	4529	14	3266	TRUE
723	1451	2783	5	1990	TRUE
724	4320	9167	38	6460	TRUE
725	76	103	0	84	TRUE
726	1529	3164	6	2261	TRUE
727	4761	10082	35	7087	TRUE
728	8536	14632	61	12236	TRUE
729	373	677	1	502	TRUE
730	67	86	0	73	TRUE

731	1015	1896	4	1387	TRUE
732	2489	4780	12	3421	TRUE
733	4407	8712	23	6181	TRUE
734	567	996	1	733	TRUE
735	1852	3476	6	2503	TRUE
736	78	115	0	92	TRUE
737	1536	2845	8	20644	TRUE
738	927	1799	4	1298	TRUE
739	1382	2723	6	1950	TRUE
740	1714	3238	6	2317	TRUE
741	4538	9268	22	6469	TRUE
742	1267	2346	8	1723	TRUE
743	3171	7074	19	5059	TRUE
744	1378	2696	7	1961	TRUE
745	2189	4198	15	3043	TRUE
746	3197	6279	17	4478	TRUE
747	2720	5322	17	3833	TRUE
748	1315	2534	10	1839	TRUE
749	2689	5322	18	3861	TRUE
750	1193	2412	7	1739	TRUE
751	1636	3071	5	2234	TRUE
752	4560	9281	21	6610	TRUE
753	1387	2592	9	1889	TRUE
755	2820	5488	21	3965	TRUE
756	3521	6939	22	4996	TRUE
763	234	617	1	395	FALSE
766	1495	2349	5	1908	TRUE
772	632	775	2	749	TRUE
773	1777	2823	5	2571	FALSE
774	1777	2823	6	2517	FALSE
775	7432	8070	32	7802	FALSE
784	940	2016	3	1502	FALSE
788	1821	2398	8	2212	TRUE
789	1913	2521	14	2329	TRUE
793	163	587	1	451	TRUE
10002	3	2	0	1	TRUE

Discussion of Results

The experimental results indicate that the performance of the algorithm is directly influenced by the structural characteristics of the graph, such as the number of nodes, the number of edges, and the number of Simple-Linear Rules generated during execution.

First, it can be observed that the execution time increases as the size of the graph grows. This increase does not depend solely on the number of nodes, but more importantly on the number of edges, since the edges determine the number of potential relationships that must be examined. As a result, two graphs with a similar number of nodes may exhibit significantly different execution times if they differ in terms of graph density.

Furthermore, the relationship between the runtime and the number of Simple-Linear Rules is particularly noteworthy. The results suggest that as the number of generated or examined rules increases, the overall execution time also increases. This behavior is expected, as each additional rule requires extra computations and validation checks. Consequently, the number of Simple-Linear Rules appears to be one of the most significant factors affecting the computational cost of the algorithm.

Despite the substantial increase in execution time for very large networks, the algorithm remains practical and effective even for graphs containing tens of thousands of nodes and edges. This demonstrates its scalability and suggests that it can be applied to realistic large-scale problems without compromising its functionality.