FernUniversität in Hagen
Faculty of Mathematics and Computer Science

AIG
Artificial Intelligence Group

Inconsistency Measurement for Classical Planning Problems

Master's Thesis

in partial fulfilment of the requirements for
the degree of *Master of Science (M.Sc.)*
in Data Science

submitted by
Sebastian Bunge

First examiner: Dr. Isabelle Kuhlmann
Artificial Intelligence Group

Advisor: Dr. Isabelle Kuhlmann
Artificial Intelligence Group

Statement

I declare that I have written the master´s thesis independently and without unauthorized use of third parties. I have only used the indicated resources and I have clearly marked the passages taken verbatim or in the sense of these resources as such. The assurance of independent work also applies to any drawings, sketches or graphical representations. The work has not previously been submitted in the same or similar form to the same or another examination authority and has not been published. By submitting the electronic version of the final version of the master´s thesis, I acknowledge that it will be checked by a plagiarism detection service to check for plagiarism and that it will be stored exclusively for examination purposes.

I explicitly agree to have this thesis published on the webpage of the artificial intelligence group and endorse its public availability.

Software created for this work has been made available as open source; a corresponding link to the sources is included in this work. The same applies to any research data.

Bonn, May 21, 2026

.....

(Place, Date) (Signature)

Zusammenfassung

Klassische Planungssysteme melden lediglich, falls ein Problem unlösbar ist, und liefern keine weiteren Erklärungen. Diese Arbeit überträgt Inkonsistenzmessung aus der Aussagenlogik auf die klassische Planung und stellt strukturelle Diagnostiken der Unlösbarkeit bereit. Eine aussagenlogische Kodierung mit dem Contension-Maß liefert uninformative Ergebnisse und motiviert Maße, die direkt auf Zielen, Operatoren und erreichbaren Zuständen operieren. Drei planungsnative Diagnoseprofile mit insgesamt sechs Maßen werden vorgeschlagen. Das Unerreichbarkeitsprofil zählt Zielpropositionen, die in keinem erreichbaren Zustand vorkommen. Das Zielmutex-Profil erkennt operatorinduzierte Ausschlüsse, bei denen Ziele einzeln erreichbar, aber nie gemeinsam erfüllbar sind. Das Zielsequenzierungskonflikt-Profil identifiziert irreversible Ausschlüsse, bei denen das Erreichen eines Ziels ein anderes dauerhaft blockiert. Jedes Profil ist durch einen Planlängenhorizont parametrisiert, wird gegen planungsadaptierte Rationalitätspostulate analysiert und ist PSPACE-vollständig. Die Profile werden mittels Answer Set Programming unter Verwendung von *plasp* und Clingo-Brave-Reasoning implementiert. Eine experimentelle Auswertung auf Benchmarks der Unsolvability Competition der International Planning Competition 2016 sowie der Eriksson-Sammlung zeigt, dass die Profile strukturell unterschiedliche Kategorien unlösbarer Instanzen unterscheiden, und offenbart ein Spannungsverhältnis zwischen Genauigkeit und Abdeckung, das der horizontbeschränkten Analyse inhärent ist.

Abstract

Classical planners report only when a problem is unsolvable, leaving the cause unexplained. This thesis adapts inconsistency measurement from propositional logic to classical planning, providing structural diagnostics of unsolvability. Encoding into propositional knowledge bases with the contension measure yields uninformative results, motivating measures that operate directly on goals, operators, and reachable states. Three planning-native diagnostic profiles comprising six measures in total are proposed. The Unreachability Profile counts goal propositions absent from all reachable states. The Goal Mutex Profile detects operator-induced mutual exclusions where goals are individually achievable but never satisfiable in a common reachable state. The Goal Sequencing Conflict Profile identifies irreversible exclusions where achieving one goal permanently blocks another. Each profile is parameterized by a plan-length horizon, analyzed against planning-adapted rationality postulates, and shown to be PSPACE-complete. The profiles are implemented in Answer Set Programming using *plasp* and Clingo brave reasoning. An experimental evaluation on the 2016 International Planning Competition's Unsolvability Competition and the Eriksson benchmark collection shows that the profiles distinguish structural categories of unsolvable instances and reveal a soundness-coverage tradeoff inherent to horizon-bounded analysis.

Contents

List of Figures

List of Tables

1. Introduction

Classical planning has become a cornerstone of automated decision-making in artificial intelligence, producing action sequences that transform an initial state into one that satisfies a goal specification. A planning problem takes as input a finite set of propositions, a set of operators with preconditions and effects over those propositions, an initial state, and a goal specification, and asks whether a finite sequence of applicable operators reaches a state in which the goal holds. Determining whether such a plan exists matters in practice. Planner runtime is wasted on unsolvable instances, model errors must be located and repaired before redeployment, and in safety-critical settings the absence of a plan signals that the operational specification needs revision. When a planning problem admits a solution, modern planners return a plan. If no solution exists, they typically report only that the problem is unsolvable. This binary failure report offers no account of why no plan exists, leaving users without the information needed to repair the problem specification or revise the modeling assumptions. Inconsistency measurement in propositional logic, surveyed by Thimm [38] and by Grant and Hunter [25], supplies a complementary line of research that quantifies the degree and structure of conflict in logical knowledge bases. Inconsistency measures assign numerical scores to knowledge bases and are characterized by rationality postulates that constrain how the score should respond to structural changes such as adding new formulas, enabling principled comparison and ranking of conflicts. Adaptations to related formalisms, including Answer Set Programming (ASP) by Ulbricht et al. [40] and temporal logic by Corea et al. [7], show that the methodology transfers beyond its original setting, yet a systematic adaptation to classical planning remains open.

Existing diagnostic work on unsolvable planning problems addresses the gap only partially. Eriksson et al. [14] provide rule-based proof systems that certify unsolvability but return binary judgments, and Göbelbecker et al. [23] introduce "excuses" that prescribe minimal repairs without characterizing the underlying conflict structure. Neither approach delivers quantitative information about how unsolvability is structured within a problem. A natural shortcut is to encode a planning problem as a propositional knowledge base and apply a classical inconsistency measure directly, but this strategy fails. For instance, the contension measure $\mathcal{I}_c$ of Grant and Hunter [24], which counts the minimum number of atoms that must take a conflicting truth value, returns a uniform value of one regardless of whether unsolvability arises from a missing operator, a pair of mutually exclusive goals, or an irreversible sequencing conflict, as shown in Section 4.2. Planning conflicts manifest as reachability and achievability failures in a state-transition system rather than as direct logical contradictions, and any faithful measure must respect this planning-specific semantics.

The goal of this thesis is to adapt inconsistency measurement to classical planning in a way that operates directly on goals, operators, and reachable states rather than reducing to propositional encodings. Three planning-native diagnostic profiles

comprising six measures in total are proposed. Each profile pairs a *scope* measure counting affected goal propositions with a *structure* measure counting structural artifacts such as unreachable propositions, mutex pairs, or sequencing-conflict pairs. The Unreachability Profile $\mathcal{I}_{\mathrm{UR}}$ counts goal propositions absent from every reachable state together with the total number of unreachable propositions, revealing cascading dependency failures. The Goal Mutex Profile $\mathcal{I}_{\mathrm{MX}}$ detects operator-induced mutual exclusions between goals that are individually achievable but never satisfiable in a common reachable state. The Goal Sequencing Conflict Profile $\mathcal{I}_{\mathrm{GS}}$ identifies irreversible exclusions in which achieving one goal permanently blocks another. Rationality postulates specify properties, such as zero-iff-solvable and monotonicity under operator or goal additions, that a measure is expected to satisfy. Each profile is parameterized by a plan-length horizon h that bounds the number of operator applications considered during reachability analysis, analyzed against planning-adapted versions of these classical postulates, and shown to be PSPACE-complete. The profiles are implemented in ASP, using *plasp* 3 by Dimopoulos et al. [9] for Planning Domain Definition Language (PDDL) translation and Clingo by Gebser et al. [19] for state-space exploration under brave reasoning. Brave reasoning is a query mode in which an atom holds if it appears in at least one answer set, so each answer set witnesses one execution trace. The implementation is evaluated on the International Planning Competition (IPC) 2016 Unsolvability Competition benchmarks by Muise and Lipovetzky [32] and the unsolvable benchmark collection by Eriksson [13].

1.1. Problem Statement

Current planning systems provide little diagnostic information when they encounter unsolvable problems. Users lack insight into why a problem has no plan and have no systematic means to distinguish or quantify structurally different causes of unsolvability. Consequently, identifying minimal changes that would restore solvability reduces to trial-and-error over the problem specification, since different types of unsolvable planning problems are neither named nor measured. This thesis supplies the missing quantitative diagnostic layer through three planning-native diagnostic profiles comprising six measures in total, targeting unreachability, operator-induced mutex, and irreversible sequencing conflicts.

1.2. Research Questions

The central research question of this thesis is:

> How can inconsistency measures from propositional logic be meaningfully adapted to classical planning to provide diagnostic value for unsolvable planning problems?

This central question is decomposed into three research questions, each addressed in a dedicated section of this thesis.

Research Question 1: Theoretical Adaptation.

> Which inconsistency measures from propositional logic can be adapted to classical planning, and what postulates do they retain?

A direct application of the contension measure to a propositional encoding of a planning problem is shown to be uninformative, motivating the development of planning-native measures. This question is answered in Section 4.

Research Question 2: Implementation.

> How can the adapted measures be efficiently computed for planning problems using Answer Set Programming?

This question is answered in Section 5, which describes the three-phase pipeline and its automated test suite.

Research Question 3: Practical Evaluation.

> Do the adapted measures provide practical diagnostic value for understanding and analyzing unsolvable planning problems?

This question is answered in Section 6, where seven compatible domains are analyzed across 158 instances.

1.3. Contributions

The contributions of this thesis are organized along the three research questions.

Theoretical Adaptation. The thesis demonstrates that naive propositional encodings of planning problems yield uninformative inconsistency measure values, with the contension measure returning one regardless of structural cause. Three planning-native diagnostic profiles comprising six measures in total are proposed and formally defined as horizon-parameterized profiles consisting of a scope measure and a structure measure: the Unreachability Profile $\mathcal{I}_{\mathrm{UR}}$, the Goal Mutex Profile $\mathcal{I}_{\mathrm{MX}}$, and the Goal Sequencing Conflict Profile $\mathcal{I}_{\mathrm{GS}}$. This thesis develops planning-adapted versions of the classical rationality postulates, namely Planning Consistency, Operator Monotonicity, and Goal Monotonicity, and analyzes all six measures against them. The containment relation between the measures is established, with sequencing conflicts implying mutex and unreachable goals excluded from both conflict analyses. The decision problems UPPER, LOWER, and EXACT for all three measures are shown to be PSPACE-complete, yielding a uniform complexity landscape that contrasts with the spread of complexity classes across measures previously observed for propositional and temporal logic adaptations.

Implementation. The measures are realized in an end-to-end ASP pipeline that accepts standard PDDL files and returns the full diagnostic profile at a user-specified horizon. The central methodological contribution is the use of brave reasoning as a decision procedure for mutex and sequencing conflicts. Different answer sets correspond to different execution traces, and the absence of a witness atom under brave reasoning constitutes a universal proof that the corresponding condition is unsatisfiable across all traces. Correctness is verified through 80 automated tests covering scenario-level validation, hierarchy relationships, library-application programming interface (API) contracts, and end-to-end PDDL processing, with all eleven test scenarios producing exact matches with their theoretically derived profiles.

Practical Evaluation. An empirical study on the IPC 2016 Unsolvability Competition and the Eriksson benchmark collection establishes the practical diagnostic value of the measures. Of 158 unsolvable instances across seven compatible domains, 68 (43%) complete within the 120-second time limit at horizon $h = 20$, with the resulting profiles distinguishing unreachability-dominated from mutex-dominated and sequencing-dominated domains. The measures provide three levels of diagnostic granularity: category classification, scope at the goal level, and structure at the relationship level. A horizon sensitivity analysis exposes a soundness-coverage tradeoff and motivates the default horizon of $h = 20$, which avoids observed false positives. The *3unsat* domain supplies the first empirical evidence of a pairwise-analysis blind spot, where instances with combinatorial unsatisfiability return zero profiles despite being unsolvable.

1.4. Thesis Structure

Section 2 establishes the theoretical foundations, covering inconsistency measurement for propositional knowledge bases, classical planning formalisms, computational complexity, and ASP. The section concludes by identifying the challenges that distinguish planning from static knowledge bases and motivate a planning-native adaptation.

Section 3 surveys the inconsistency measurement literature and positions the thesis against existing work on planning diagnostics, including dead-state certificates [14], repair-based excuses [23], and explanation methods for unsolvable tasks [37]. It also discusses adaptations of inconsistency measurement to non-propositional domains, which serve as methodological precedents.

Section 4 develops the three planning-native measures. It opens with a structural taxonomy of unsolvable planning problems, demonstrates the limitations of naive propositional encodings on six running test scenarios, introduces horizon-bounded reachability, and formally defines the Unreachability, Goal Mutex, and Goal Sequencing Conflict Profiles. The section then analyzes the measures against rationality postulates, establishes formal relationships between them, and proves uniform PSPACE-completeness of their decision problems.

Section 5 presents the ASP-based implementation. It describes the three-phase pipeline, the adoption of *plasp* for PDDL translation, the reachability and witness-collection encodings used under Clingo brave reasoning, the Python module that extracts measure values, and the automated test suite that validates the pipeline.

Section 6 reports the experimental evaluation. It describes the methodology, presents per-domain results across the seven compatible domains, discusses case studies that illustrate the diagnostic capabilities and the pairwise-analysis limitation, analyzes horizon sensitivity, and examines computational performance.

Section 7 summarizes the contributions along the three research questions, discusses limitations, provides a closing assessment of the methodological insight that planning conflicts manifest as reachability and achievability failures rather than direct logical contradictions, and identifies directions for future work.

2. Foundations

This section establishes the theoretical foundations for adapting inconsistency measurement to classical planning problems. The section begins with formal definitions for knowledge bases and inconsistency measurement, covering both syntactic and semantic approaches. This is followed by an examination of classical planning formalisms and the specific challenges that arise when adapting inconsistency measurement to the planning domain.

2.1. Inconsistency Measurement

Thimm [38] presents inconsistency measurement as a formal method for quantifying the degree of inconsistency within logical knowledge bases. Rather than treating inconsistency as a binary property, these measures assign numerical values that reflect the severity or extent of conflicts, enabling comparative analysis and systematic approaches to conflict resolution.

2.1.1. Knowledge Bases and Inconsistency Measures

Definition 1 (Knowledge Base [38])**.** A *knowledge base* K is a finite set of propositional formulas. Let $\mathbb{K}$ denote the set of all finite propositional knowledge bases. A knowledge base K is *consistent* if there exists a truth assignment satisfying all formulas in K; otherwise it is *inconsistent*.

Example 1 (Consistent and Inconsistent Knowledge Bases)**.** The knowledge base $K_1 = \{a, a \rightarrow b\}$ is consistent. The truth assignment $a = b =$ true satisfies both formulas. The knowledge base $K_2 = \{a, \neg a\}$ is inconsistent since no truth assignment can satisfy both formulas.

An inconsistency measure provides a systematic way to assign numerical values reflecting the degree of inconsistency.

Definition 2 (Inconsistency Measure [38])**.** An *inconsistency measure* is a function $\mathcal{I} : \mathbb{K} \to \mathbb{R}_{\geq 0}$ that maps each knowledge base to a non-negative real number quantifying its degree of inconsistency.

Inconsistency measures are typically required to satisfy certain rationality postulates that capture intuitive properties of inconsistency quantification. Hunter and Konieczny [27] and Thimm [38] identify the following core postulates:

1. **Consistency:** $\mathcal{I}(K) = 0$ if and only if K is consistent. This ensures the measure correctly identifies the absence of conflicts.
2. **Monotonicity:** If $K \subseteq K'$, then $\mathcal{I}(K) \leq \mathcal{I}(K')$. Adding formulas cannot decrease inconsistency in classical propositional logic.
3. **Free Formula Independence:** $\mathcal{I}(K \cup \{\phi\}) = \mathcal{I}(K)$ if ϕ is not involved in any inconsistency of $K \cup \{\phi\}$. Irrelevant formulas do not affect the measure.

Example 2 (Rationality Postulates)**.** Consider $K = \{a, \neg a\}$. By Consistency, $\mathcal{I}(K) > 0$ since K is inconsistent. Adding the formula b yields $K' = \{a, \neg a, b\}$; by Monotonicity, $\mathcal{I}(K') \geq \mathcal{I}(K)$. Since b shares no atoms with the conflict and is therefore not involved in any inconsistency, Free Formula Independence implies $\mathcal{I}(K') = \mathcal{I}(K)$.

These postulates may require modification when adapting measures to specific domains. In particular, Monotonicity assumes that adding information can only increase inconsistency, which does not hold in non-monotonic contexts where new information can resolve conflicts.

2.1.2. The Contension Measure

Several approaches to inconsistency measurement exist, including syntactic methods based on minimal inconsistent subsets and semantic methods based on model-theoretic analysis [38]. Among semantic methods, the contension measure $\mathcal{I}_c$ introduced by Grant and Hunter [24] is particularly relevant as a baseline for domain adaptations.

The contension measure employs three-valued paraconsistent semantics based on Priest's Logic of Paradox [35], where propositions can be assigned a third truth value B ("both true and false") representing conflict. The measure counts the minimum number of *atoms* that must be assigned B to satisfy the knowledge base under three-valued semantics:

$$\mathcal{I}_c(K) = \min\{|\upsilon^{-1}(B) \cap \text{At}(K)| \mid \upsilon \models_3 K\}$$

where $\upsilon : \text{At}(K) \to \{T, F, B\}$ is a three-valued truth assignment, $\text{At}(K)$ denotes the atoms appearing in K, and $\upsilon \models_3 K$ indicates that υ satisfies K under three-valued semantics.

Example 3 (Contension Measure)**.** Consider $K = \{a \wedge b, \neg a, \neg b\}$. Setting both a and b to B satisfies all formulas: $a \wedge b$ evaluates to B under the three-valued semantics where conjunction takes the minimum value with ordering $F < B < T$, and $\neg a$ and $\neg b$ both evaluate to B (negation of B is B). Since the truth value B propagates through negation, this is the minimal resolution, yielding $\mathcal{I}_c(K) = 2$.

The contension measure counts atoms, not formulas or conflicts. This means that multiple formula conflicts may be resolved by assigning B to a single shared atom. For instance, the knowledge base $K = \{a \wedge b, \neg a \wedge b, \neg a \wedge c, a \wedge c\}$ contains multiple pairwise conflicts but can be satisfied by setting only a to B, yielding $\mathcal{I}_c(K) = 1$. This atom-level granularity distinguishes contension from formula-counting measures.

Section 4.2 demonstrates that despite this granularity, $\mathcal{I}_c$ yields uninformative results when applied to naive propositional encodings of planning problems. Domain adaptations of inconsistency measurement to non-classical settings, including ASP [40] and temporal logic for declarative processes [7], are discussed in Section 3.

2.2. Classical Planning

Russell and Norvig [36] describe classical planning as operating on factored representations, meaning the world state is decomposed into individual variables like "robot at location A" or "door is open" rather than treating each complete world situation as an indivisible atomic state. The formal definitions that follow adopt the propositional Stanford Research Institute Problem Solver (STRIPS) formulation of Bylander [6], which also serves as the reference for the later complexity results. These representations are built over a finite set P of atomic propositions. The following definitions introduce states and operators over P, then assemble them into the formal notion of a planning problem.

Definition 3 (State [36])**.** A state s is a subset $s \subseteq P$ representing the propositions that are true in that state.

States follow database semantics with two key assumptions [36, 6]. The *closed-world assumption* treats unmentioned propositions as false. Any proposition $p \in P$ not explicitly in s is considered false in state s, eliminating the need to explicitly represent negative facts. The *unique names assumption* ensures that distinct object names refer to distinct entities in the domain.

Definition 4 (Operator [6])**.** An operator o is a triple $\langle \text{precond}(o), \text{add}(o), \text{delete}(o) \rangle$ where $\text{precond}(o) \subseteq P$ are the preconditions, $\text{add}(o) \subseteq P$ are the add effects, and $\text{delete}(o) \subseteq P$ are the delete effects.

Operators are ground, meaning all variables have been replaced by specific objects [6].

Definition 5 (Planning Problem [6])**.** A planning problem is a 4-tuple $\langle P, O, I, G \rangle$ where P is a finite set of atomic propositions, O is a finite set of operators, $I \subseteq P$ is the initial state, and $G \subseteq P$ is the goal specification.

Definition 6 (Applicability and Result [6])**.** An operator o is applicable in state s if and only if $\text{precond}(o) \subseteq s$. The result of applying o in s is:

$$\text{RESULT}(s, o) = (s \setminus \text{delete}(o)) \cup \text{add}(o)$$

Example 4 (Operator Application)**.** Consider state $s = \{a, b\}$ and operator $o = \langle \{a\}, \{c\}, \{b\} \rangle$ with $\text{precond}(o) = \{a\}$, $\text{add}(o) = \{c\}$, and $\text{delete}(o) = \{b\}$. Since $\{a\} \subseteq \{a, b\}$, operator o is applicable in s. The result is $\text{RESULT}(s, o) = (\{a, b\} \setminus \{b\}) \cup \{c\} = \{a, c\}$.

Definition 7 (Plan [36])**.** A plan is a sequence of operators $\langle o_1, o_2, \ldots, o_k \rangle$ such that each o_i is applicable in state s_{i-1}, where $s_0 = I$ and $s_i = \text{RESULT}(s_{i-1}, o_i)$, and the final state satisfies $G \subseteq s_k$.

Definition 8 (Solvability [6])**.** A planning problem $\langle P, O, I, G \rangle$ is solvable if there exists a plan and unsolvable otherwise.

Determining whether a planning problem is solvable is PSPACE-complete [6], which has motivated extensive research into efficient algorithms and search techniques.

While the preceding definitions establish the formal semantics of planning, practical planning systems require a concrete specification language. The PDDL, introduced by Ghallab et al. [21] for the 1998 International Conference on Artificial Intelligence Planning Systems competition, was developed to provide a standardized language for describing planning problems. PDDL has since become the de facto standard for representing classical planning problems, evolving through multiple versions to support increasingly expressive features while maintaining a foundational core based on the STRIPS representation of Fikes and Nilsson [16]. The STRIPS framework established the foundational principles for modeling planning domains where actions have deterministic effects and the world is fully observable, which PDDL extends with additional expressiveness and standardization.

Throughout this thesis, the term "propositional PDDL" refers to PDDL specifications using the `:strips` and `:typing` requirements without numeric fluents or other extensions from PDDL 2.1 and later versions. The `:strips` requirement indicates the use of basic precondition-effect actions following the classical STRIPS model, while `:typing` enables hierarchical type declarations for objects, allowing parameters to be restricted to specific types. Another common requirement is `:equality`, which enables the use of equality predicates to compare object references. This thesis focuses on `:strips` and `:typing`, corresponding to the classical planning model with propositional state representations, typed objects, and deterministic actions with conjunctive preconditions and effects.

PDDL separates planning specifications into two complementary files: a domain file that defines the reusable structure of a planning domain, and a problem file that specifies a particular instance within that domain. This separation enables domain definitions to be reused across multiple problem instances, similar to the distinction between schema and data in database systems.

A domain file specifies the abstract structure using the following components:

$$\text{Domain} = \langle \text{Requirements}, \text{Predicates}, \text{Actions} \rangle$$

where Requirements declares which PDDL features are used (such as `:strips`, `:typing`, `:equality`), Predicates defines the vocabulary of Boolean properties, and Actions specifies the available operators.

A problem file instantiates the domain with concrete elements:

$$\text{Problem} = \langle \text{DomainRef}, \text{Objects}, \text{Init}, \text{Goal} \rangle$$

where DomainRef references the domain name, Objects enumerates the entities in this instance, Init specifies the initial state, and Goal defines the goal specification. For instance, a logistics domain file would define predicates like $\text{at}(x, y)$ and actions like drive, while a problem file would specify particular trucks, locations, and delivery goals.

Predicates define the vocabulary of a planning domain as parameterized Boolean functions. Formally, a predicate declaration has the form:

$$\text{pred}(x_1, x_2, \ldots, x_n)$$

where pred is the predicate name and $x_1, \ldots, x_n$ are typed or untyped parameters. When instantiated with concrete objects, a predicate becomes a ground atom (proposition) that can be true or false in a state.

For example, in the blocksworld domain (Example 5), the predicate $\text{on}(x, y)$ with parameters x and y representing blocks becomes the ground atom $\text{on}(a, b)$ when instantiated with specific blocks a and b, representing "block a is on block b." The set of all ground atoms formed from domain predicates and problem objects constitutes the proposition set P in the planning problem formalization.

PDDL actions define parameterized operators using a precondition-effect structure. An action schema has the form:

$$\text{action}(\text{params}) = \langle \text{precond}(\text{params}), \text{eff}(\text{params}) \rangle$$

where params are typed parameters, $\text{precond}(\text{params})$ is a logical formula specifying preconditions, and $\text{eff}(\text{params})$ specifies effects as a conjunction of positive and negative literals.

In basic PDDL, preconditions are conjunctions of positive atoms, while effects are conjunctions of positive and negated atoms. The set-based operator representation directly captures this structure:

$$\begin{aligned} \text{precond}(o) &= \{p \mid p \text{ appears positively in the precondition of } o\} \\ \text{add}(o) &= \{p \mid p \text{ appears positively in the effect of } o\} \\ \text{delete}(o) &= \{p \mid \neg p \text{ appears in the effect of } o\} \end{aligned}$$

Ground operators result from instantiating action parameters with specific objects from the problem definition, yielding the finite operator set O in the planning problem tuple.

The initial state in PDDL is specified as a conjunction of ground atoms:

$$\text{Init} = \bigwedge_{i=1}^{n} p_i$$

which corresponds to the set $I = \{p_1, p_2, \ldots, p_n\} \subseteq P$ under the closed-world assumption where unmentioned propositions are false.

Goal specifications in PDDL can be arbitrary logical formulas in the full language, but in basic PDDL with `:strips` requirements they are restricted to conjunctions of positive atoms. Formally, a PDDL goal $\text{Goal} = g_1 \wedge g_2 \wedge \cdots \wedge g_m$ corresponds to the goal set $G = \{g_1, g_2, \ldots, g_m\} \subseteq P$ in the planning problem formalization. A state s satisfies the goal if and only if $G \subseteq s$. Note that negative goal literals (e.g., $\neg p$) require the PDDL 3.0 extensions of Gerevini and Long [20]. This thesis focuses on positive goal conjunctions.

While PDDL has evolved through multiple versions, PDDL 2.1 [18] added numeric fluents and durative actions, PDDL 2.2 [11] introduced derived predicates and timed initial literals, PDDL 3.0 [20] added preferences and constraints, and PDDL+ [17] supports continuous processes. This thesis focuses on basic forms of PDDL due to their foundational simplicity and well-understood semantics. Simple PDDL formulations with conjunctive preconditions and effects provide sufficient expressiveness for systematic adaptation of inconsistency measurement without the complications of temporal reasoning, complex preferences, or soft constraints.

Example 5 (Blocksworld Domain)**.** Consider a simplified blocksworld domain with predicates $\text{on}(x, y)$, $\text{ontable}(x)$, $\text{clear}(x)$, $\text{holding}(x)$, and handempty. The action for stacking block x on block y is defined as:

$$\begin{aligned} &\text{action stack}(x, y) \\ &\quad \text{precond} : \text{holding}(x) \wedge \text{clear}(y) \\ &\quad \text{eff} : \text{on}(x, y) \wedge \text{clear}(x) \wedge \text{handempty} \\ &\qquad\qquad \wedge \neg\text{holding}(x) \wedge \neg\text{clear}(y) \end{aligned}$$

When instantiated with specific blocks a and b from a problem file, this yields the ground operator:

$$\begin{aligned}\text{stack}(a, b) = \langle &\{\text{holding}(a), \text{clear}(b)\}, \\ &\{\text{on}(a, b), \text{clear}(a), \text{handempty}\}, \\ &\{\text{holding}(a), \text{clear}(b)\}\rangle\end{aligned}$$

demonstrating how PDDL action schemas correspond to the set-based operator representation.

This formalization of PDDL provides the necessary foundation for discussing how planning problems are specified in practice, while the earlier formal planning definitions establish the semantic interpretation of these specifications as state spaces and reachability problems.

Modern planning systems employ various search strategies to explore different ways of reaching goals, such as searching forward from the initial state, working backward from the goal, or examining the space of possible plans [22, 36]. These strategies include techniques that build graphical representations of planning problems to identify conflicts and estimate solution difficulty, such as the Graphplan algorithm of Blum and Furst [3], and heuristic search methods developed by Bonet and Geffner [4] and Hoffmann and Nebel [26] that use simplified versions of problems to guide their search toward solutions. These heuristic approaches have significantly improved planning performance for solvable instances, but when these systems encounter an unsolvable problem, as observed by Eriksson et al. [14], they typically provide little diagnostic information beyond reporting failure. This diagnostic gap represents a significant limitation for users attempting to understand the source of conflicts or determine appropriate modifications to achieve solvability.

2.3. Computational Complexity

This section introduces the complexity-theoretic concepts used in the analysis of the proposed measures. All definitions and results follow Papadimitriou [34].

A *complexity class* groups decision problems by the computational resources required to solve them. The two fundamental resources are time, measured in computation steps, and space, measured in memory cells used. Each can be bounded under deterministic or nondeterministic computation. A complexity class is therefore specified by a mode of computation, a resource, and a bounding function $f(n)$ on the resource usage as a function of input size n.

Definition 9 (Space Complexity Classes [34])**.** Let $f : \mathbb{N} \to \mathbb{N}$ be a function.

- SPACE($f(n)$) is the class of languages decidable by a deterministic Turing machine using at most $f(n)$ cells of working memory on inputs of size n.
- NSPACE($f(n)$) is the class of languages decidable by a nondeterministic Turing machine using at most $f(n)$ cells of working memory on inputs of size n.

The parametrized classes that collect all problems solvable in polynomial space are defined as:

$$\text{PSPACE} = \bigcup_{k \geq 0} \text{SPACE}(n^k) \qquad \text{NPSPACE} = \bigcup_{k \geq 0} \text{NSPACE}(n^k)$$

for deterministic and nondeterministic computation, respectively.

A key distinction between time and space complexity is that space can be reused. A computation may visit the same memory cells repeatedly across different stages of execution, whereas time steps, once spent, cannot be recovered.

Theorem 1 (Savitch's Theorem [34])**.** *For any function $f(n) \geq \log n$, NSPACE$(f(n)) \subseteq$ SPACE$(f(n)^2)$.*

The central idea behind Savitch's theorem is a "middle-first search" strategy. To determine whether a configuration C_1 can reach another configuration C_2 within 2^i steps, the algorithm guesses a midpoint configuration C_m and recursively checks whether C_1 reaches C_m within 2^{i-1} steps and C_m reaches C_2 within 2^{i-1} steps. Since the recursion depth is at most $O(f(n))$ and each recursive call stores only a constant number of configurations of size $O(f(n))$, the total space used is $O(f(n)^2)$.

An immediate consequence of Savitch's theorem is that PSPACE = NPSPACE, meaning nondeterminism does not increase the power of polynomial-space computation. This stands in contrast to the relationship between P and NP, where nondeterminism is widely considered to provide additional power.

Definition 10 (Complement and co-PSPACE [34])**.** For a complexity class C, the complement class co-C is the class $\{L : \bar{L} \in C\}$, where $\bar{L}$ denotes the complement of language L. Deterministic complexity classes are closed under complement, since a deterministic machine deciding L can be converted to one deciding $\bar{L}$ by swapping acceptance and rejection. In particular, PSPACE = co-PSPACE.

The closure under complement has an important structural consequence. For problems in PSPACE, the distinctions between a problem and its complement, between NP-type and coNP-type formulations, collapse. The same machine that decides membership also decides non-membership, and the Boolean combination of polynomially many PSPACE problems remains in PSPACE.

Definition 11 (PSPACE-completeness [34])**.** A language L is *PSPACE-hard* if every language $L' \in$ PSPACE is polynomial-time reducible to L, written $L' \leq_p L$. A language L is *PSPACE-complete* if $L \in$ PSPACE and L is PSPACE-hard. A completeness proof consists of two parts: *membership* shows $L \in$ PSPACE, and *hardness* shows $L' \leq_p L$ for some known PSPACE-hard language L'.

PSPACE-complete problems represent the hardest problems in PSPACE, and solving any one of them in polynomial time would imply P = PSPACE. The canonical PSPACE-complete problem is quantified Boolean satisfiability (QSAT), which asks

whether a fully quantified Boolean formula with alternating universal and existential quantifiers evaluates to true [34].

For function problems that ask for a computed value rather than a yes-or-no answer, the corresponding complexity class is FPSPACE, the class of functions computable by a deterministic Turing machine using polynomial working space. Binary search with polynomially many calls to a PSPACE decision oracle yields an FPSPACE computation, since PSPACE is closed under polynomial composition.

2.4. Answer Set Programming

Dimopoulos et al. [10] describe ASP as a declarative programming paradigm based on stable model semantics for logic programs. This section provides formal definitions of ASP concepts used in the implementation section.

Definition 12 (Logic Program [10])**.** A *logic program* Π is a finite set of rules. Each rule has the form:

$$h \leftarrow b_1, \ldots, b_m, \mathbf{not}\ c_1, \ldots, \mathbf{not}\ c_n$$

where h is the *head* (an atom or empty), $b_1, \ldots, b_m$ are the *positive body* atoms, and $c_1, \ldots, c_n$ are the *negative body* atoms preceded by default negation **not**.

A rule with an empty head is a *constraint* (denoted $\leftarrow b_1, \ldots, b_m$), which eliminates answer sets containing the body atoms. A rule with an empty body is a *fact* (denoted $h.$).

Definition 13 (Answer Set [10])**.** An *answer set* (or *stable model*) of a logic program Π is a minimal set of atoms S such that:

1. For every rule $h \leftarrow b_1, \ldots, b_m, \mathbf{not}\ c_1, \ldots, \mathbf{not}\ c_n$ in Π: if $\{b_1, \ldots, b_m\} \subseteq S$ and $\{c_1, \ldots, c_n\} \cap S = \emptyset$, then $h \in S$.
2. S is minimal with respect to set inclusion among all sets satisfying condition 1.

Example 6 (Simple ASP Program)**.** Consider the program with rules: $a.$ (fact), $b \leftarrow a.$ (conditional), $\leftarrow a, c.$ (constraint). The unique answer set is $\{a, b\}$: the fact establishes a, the conditional rule derives b from a, and c is not derived so the constraint is not violated.

Eiter and Gottlob [12] formalize two complementary modes of reasoning over the answer sets of a logic program.

Definition 14 (Brave and Cautious Consequences [12])**.** Let Π be a logic program with answer sets $\text{AS}(\Pi) = \{S_1, \ldots, S_n\}$. The *brave consequences* and *cautious consequences* of Π are defined as:

$$\text{BRAVE}(\Pi) = \bigcup_{S \in \text{AS}(\Pi)} S \qquad \text{(atoms true in at least one answer set)}$$

$$\text{CAUTIOUS}(\Pi) = \bigcap_{S \in \text{AS}(\Pi)} S \qquad \text{(atoms true in every answer set)}$$

Modern ASP solvers such as Clingo, developed by Gebser et al. [19], support both modes natively. In a planning context with non-deterministic operator selection, each answer set corresponds to a different execution trace. Brave consequences capture everything achievable under any choice of operators, making brave reasoning the appropriate mode for reachability analysis.

ASP provides a natural framework for encoding planning problems. The modern tool *plasp* 3 by Dimopoulos et al. [9] translates PDDL specifications into ASP programs where answer sets correspond to valid plans. The system generates rules encoding operator applicability, effect application, and goal satisfaction.

Definition 15 (Sequential Planning Encoding [9])**.** A *sequential encoding* for horizon i, denoted $S(i)$, is an ASP program that searches for plans of exactly i steps. The encoding includes time-indexed fluents $\text{holds}(p, t)$ for propositions p at time steps $t \in \{0, 1, \ldots, i\}$, action atoms $\text{occurs}(o, t)$ indicating operator o executes at step t, and constraints ensuring exactly one action occurs at each step.

Sequential encodings enforce that exactly one operator executes at each time step. To find plans of unknown length, the encoding is solved iteratively for increasing horizons $i = 0, 1, 2, \ldots$ until either an answer set is found or a termination condition is met.

Parallel encodings relax the single-action restriction, allowing multiple non-interfering actions to execute in parallel at each step. To ensure valid parallel execution, these encodings include *serialization constraints* that prevent conflicting actions from occurring at the same time step. Two actions conflict if one deletes a precondition or add effect of the other, or if both add and delete the same proposition. Serialization constraints take the form $\leftarrow \text{occurs}(o_1, t), \text{occurs}(o_2, t)$ for each pair of conflicting actions o_1 and o_2.

For inconsistency measurement applications, ASP offers several advantages. First, the declarative nature of ASP programs allows for systematic analysis of conflicts through unsatisfiable cores and related computational structures, as demonstrated by Ulbricht et al. [40] in their adaptation of inconsistency measures to ASP. Second, ASP provides natural support for non-monotonic reasoning through default negation, which is essential for planning domains where the monotonicity postulate of classical inconsistency measures may not hold due to the frame problem.

Regarding computational performance, the choice between ASP and Boolean Satisfiability (SAT)-based approaches depends on the specific measures and encodings.

Kuhlmann et al. [29] found that ASP-based algorithms outperform basic SAT encodings for several inconsistency measures. However, Niskanen et al. [33] demonstrated that specialized MaxSAT approaches can significantly outperform ASP methods for the same measures. This thesis employs ASP for implementation due to its expressiveness for encoding reachability-based measures and its natural support for the non-monotonic reasoning patterns that arise in planning diagnostics.

2.5. Challenges in Adapting Inconsistency Measures to Planning

While inconsistency measures have been successfully adapted to temporal domains, planning presents unique challenges that distinguish it from static knowledge bases and temporal constraint problems. Unlike propositional knowledge bases where inconsistency arises from contradicting formulas, planning inconsistency emerges from the interaction between initial states, action sequences, and goal specifications within a dynamic system.

The fundamental challenge lies in the temporal and causal nature of planning. In classical inconsistency measurement, conflicts are typically identified among static formulas through direct logical contradiction. In planning, however, conflicts arise through complex interactions: action preconditions that cannot be satisfied given available resources, goal states that are mutually exclusive, or action sequences that lead to dead-end states from which no goal can be reached. These conflicts manifest not as direct logical contradictions but as reachability problems in the state space defined by the planning domain.

Furthermore, planning domains involve the frame problem, where unmentioned propositions are assumed to persist unchanged unless explicitly affected by actions. This introduces default assumptions that complicate the direct application of traditional inconsistency measures, which assume explicit representation of all relevant information. The closed-world assumption in planning domains means that conflicts may arise from the absence of enabling conditions rather than the presence of contradictory assertions.

Adapting inconsistency measures to planning requires addressing how conflicts manifest in the context of action sequences, state transitions, and goal reachability. Unlike temporal constraint problems where constraints operate on continuous variables with explicit temporal intervals, planning involves discrete state transitions governed by preconditions and effects.

These challenges have significant implications for measure selection. Paraconsistent approaches such as the contension measure, which use three-valued semantics where propositions can be assigned B ("both true and false") to represent conflicts, can theoretically be applied to planning encodings just as they are applied to propositional knowledge bases. However, as demonstrated in Section 4.2, naive propositional encodings of planning problems yield uninformative measure values that fail to distinguish structurally different failures. The contension measure returns the same value for problems with fundamentally different structural causes of

unsolvability, providing no diagnostic insight. This motivates the development of planning-native measures that operate directly on reachability properties, capturing the structural relationships between goals, operators, and state transitions that give rise to planning-specific conflicts. Before developing these measures, Section 3 reviews how prior work has approached inconsistency and unsolvability in planning and related formalisms.

3. Related Work

With the formal foundations and the planning-specific conflict challenges now established, this section situates the thesis within existing research on inconsistency measurement and planning diagnostics. The connection between inconsistency and planning has been explored from several perspectives, with foundational work establishing important theoretical and diagnostic approaches for unsolvable planning problems. Thimm [38] and Grant and Hunter [25] survey the broader field of inconsistency measurement in propositional logic, where various techniques quantify conflicts in logical knowledge bases. Their systematic adaptation to planning domains remains an open research direction. Recent work by Kuhlmann et al. [30] applies inconsistency metrics to diagnose non-executable plans in robotics and cognitive systems, demonstrating the emerging interest in bridging these fields.

Eriksson et al. [14] establish important theoretical groundwork on proof systems for demonstrating unsolvability in planning. This work introduces the concept of "dead states," defined as states from which no goal state is reachable, and proposes a rule-based proof system for certifying unsolvability. This concept connects directly to the sequencing conflict measure proposed in this thesis. A sequencing conflict between goals g_1 and g_2 exists precisely when all reachable states satisfying g_1 are dead states for g_2. While Eriksson et al. [14] provide binary certificates (solvable/unsolvable), the proposed measure quantifies the extent of such conflicts, revealing how many goal pairs exhibit sequencing conflicts and whether conflicts are unidirectional or symmetric.

From a diagnostic perspective, Göbelbecker et al. [23] introduced "excuses" for unsolvable planning tasks: minimal modifications required to restore solvability, such as adding initial facts, introducing operators, or weakening goals. An "add producer" excuse introduces an operator whose effects include a missing proposition, while an "add reversal operator" excuse introduces an operator that undoes an irreversible action. While excuses provide actionable repair suggestions, the measures proposed here complement this by providing quantitative structural information without prescribing specific repairs. The unreachability measure identifies which propositions cannot be achieved, suggesting where add-producer excuses are needed. The sequencing conflict measure identifies irreversible action sequences, suggesting where add-reversal excuses apply. In keeping with the descriptive orientation of inconsistency measurement, which characterizes the conflicts present in a knowledge base rather than prescribing how they should be resolved, the pro-

posed measures report objective structural properties and leave repair decisions to domain experts.

The planning literature has also extensively studied mutex (mutual exclusion) relationships, particularly in the context of the Graphplan algorithm of Blum and Furst [3]. Classical mutex analysis identifies proposition pairs that cannot both be true at a given planning level, which are used heuristically during search. The mutex measure proposed in this thesis extends this concept to goal-level diagnostics, identifying goals that are each individually achievable but never co-occur in any reachable state, quantifying both scope, the number of involved goals, and structure, the number of mutex pairs.

Complementing these approaches, Sreedharan et al. [37] developed methods for producing explanations that help users understand why planning tasks are unsolvable, focusing on the identification of unreachable goal propositions. While valuable for understanding individual cases, these explanation methods do not provide the quantitative measures needed for systematic comparison across different unsolvable instances.

The adaptation of inconsistency measures to non-propositional domains provides important methodological precedents for this thesis. Ulbricht et al. [40] adapted inconsistency measures to ASP, demonstrating that non-monotonic reasoning requires reformulated rationality postulates. Their key observation, that adding information to a non-monotonic knowledge base can reduce inconsistency, directly parallels the planning context, where adding operators can restore solvability. This motivates the Operator Monotonicity postulate (Section 4.5.1), which reverses the direction of classical monotonicity.

Corea et al. [7] developed inconsistency measures for declarative process specifications expressed in temporal logic, addressing a domain with structural parallels to planning. Their work identifies limitations of direct propositional measure application when temporal operators and activity orderings are involved, similar to the finding in this thesis that naive propositional encoding of planning problems yields uninformative measure values (Section 4.2). Both domains involve sequential constraint satisfaction over time, though planning operates on state transitions while process specifications operate on activity traces. Corea et al.'s approach of developing domain-specific semantics for conflict detection, rather than reducing to propositional logic, validates the strategy of defining planning-native measures that operate directly on goals, operators, and reachability.

More broadly, the inconsistency measurement literature has established that different domains require different notions of what constitutes a "conflict." In propositional logic, conflicts are direct contradictions. In temporal logic, conflicts involve unsatisfiable constraint combinations over future states. In planning, conflicts manifest as unreachable propositions, mutex goal pairs, or irreversible sequencing. Each domain adaptation preserves the core methodology, namely formal definitions, rationality postulates, and complexity analysis, while instantiating domain-appropriate semantics.

These research efforts establish important foundations for understanding unsolvable planning problems. However, they share a common limitation: a focus on binary characterizations or prescriptive repairs rather than quantitative structural analysis. This thesis bridges the gap by adapting inconsistency measurement methodology to provide systematic quantitative diagnostics for planning problems. Section 4 develops this adaptation, defining the three planning-native diagnostic profiles and analyzing their formal properties.

4. Theoretical Adaptation

This section presents three proposals for adapting inconsistency measurement to classical planning problems. This work develops planning-native measures that operate directly on planning structures: goals, operators, and precondition-effect relationships. The approach employs multi-faceted diagnostic profiles rather than single severity scores, providing actionable information for problem repair without making assumptions about which repairs are valid.

The section begins with a structural taxonomy for unsolvable planning problems (Section 4.1), followed by motivation for planning-native measures (Section 4.2) and six test scenarios (Section 4.3) that serve as running examples. Section 4.4 presents the three measure proposals. Section 4.5 analyzes rationality postulates and computational complexity. Section 4.6 discusses diagnostic interpretation and normalization considerations.

4.1. Structural Taxonomy for Unsolvable Planning Problems

Before presenting the proposed measures, this section establishes a structural taxonomy (Figure 1) that organizes unsolvable planning problems by features relevant to inconsistency measurement. This taxonomy identifies distinct measurement targets and motivates the design choices in subsequent sections.

4.1.1. Structural Categories

Unsolvable planning problems can be organized by structural features that suggest different approaches for inconsistency measurement. The taxonomy is organized by the analytical approach needed to detect and quantify each kind of conflict. The categories overlap rather than partition, and a single planning problem may exhibit features from more than one. The three structural categories are:

Category 1: Operator-Induced Goal Conflicts. The goal specification G contains propositions that cannot all be true in any valid state. In classical propositional logic, this would include explicit contradictions such as $\{p, \neg p\}$. However, in propositional PDDL with `:strips` requirements, goal specifications consist exclusively of positive literals. Negative goal literals require PDDL 3.0 extensions [20]. Con-

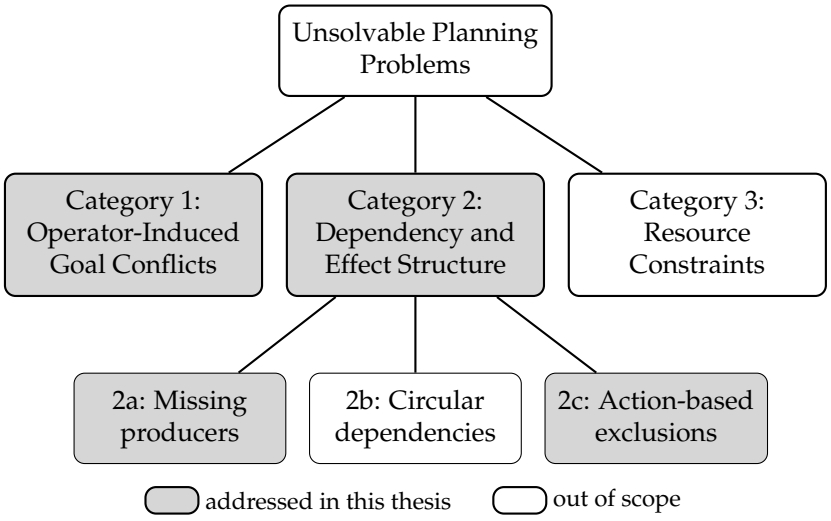

Figure 1: Structural taxonomy for unsolvable planning problems.

sequently, specification-level contradictions of the form $\{p, \neg p\}$ cannot occur in the planning formalism addressed by this thesis.

Nevertheless, goal propositions may be *operator-induced mutex* (mutual exclusion). Each proposition is individually achievable, but action effects structurally prevent them from holding in the same reachable state. For instance, a goal requiring both $\{light\_on, light\_off\}$ when operators enforce mutual exclusion, where the turn-on operator deletes *light_off* and turn-off deletes *light_on*, cannot be satisfied by any reachable state. This operator-induced mutex represents the planning analog of specification-level conflicts. This category is detectable through reachability analysis of G against the state space induced by O, and suggests measures that count goal propositions involved in mutex relationships and quantify the density of those relationships across the goal set.

Category 2: Dependency and Effect Structure. Problems arise in the precondition, add-effect, and delete-effect relationships between propositions and operators. The *precondition-effect dependency graph* captures preconditions and add effects. Nodes represent propositions, and directed edges connect preconditions to effects through operators, indicating that achieving a proposition p requires first satisfying proposition q. For example, if operator o has $\text{precond}(o) = \{q\}$ and $\text{add}(o) = \{p\}$, the graph contains an edge from q to p labeled with o. Figure 2 illustrates this for the Locked Door scenario (Scenario 1 in Section 4.3), where the proposition *has_key* has no producer, making it the root cause of unreachability. Delete effects are not captured by this graph and are addressed separately in subtype 2c.

This category encompasses three structural features. The first two are analyzable via graph algorithms on the precondition-effect dependency graph, while the third

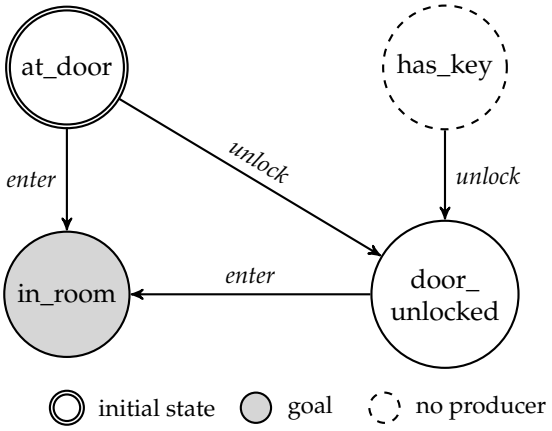

Figure 2: Precondition-effect dependency graph for the Locked Door scenario.

extends to delete effects and requires trajectory analysis of how operator applications irreversibly alter reachable states:

- **(2a) Missing producers:** No operator produces a required proposition. In the dependency graph, a goal proposition has no incoming edges. For example, unlocking a door when no operator can produce the required key.
- **(2b) Circular dependencies:** Preconditions form cycles without initialization. In the dependency graph, a cycle exists where no node in the cycle is initially true. For example, requiring a security badge to be activated to enter a building while needing to be inside the building to activate the badge.
- **(2c) Action-based exclusions:** Achieving certain goal propositions via irreversible actions creates dead-end states preventing other goal propositions from being reached. For example, entering a one-way door to retrieve an item while becoming trapped and unable to complete other objectives.

This category suggests reachability-based measures that count blocked goal propositions and trace their cascading dependency failures. Cycle detection and producer counting are natural extensions left to future work.

Category 3: Resource Constraints. Consumable resources such as time, fuel, budget, or battery are insufficient to satisfy goal requirements. This includes both total resource deficit, where the cumulative consumption exceeds the available supply, as when two fuel-consuming moves are required but only one unit of fuel is available, and resource contention, where multiple goal propositions compete for the same limited resource, as when two surfaces require painting but only one can of paint is available. This category requires numeric tracking and accounting measures, such as computing absolute or relative resource deficits, counting over-subscribed resources, or tracking path-dependent consumption patterns.

As an example of category overlap, resource contention problems in Category 3 often exhibit operator-induced goal conflicts from Category 1 caused by resource consumption. The taxonomy is organized by what measurement approaches are applicable: logical analysis for operator-induced goal conflicts, graph and trajectory analysis for dependency and effect structure, and numeric accounting for resources.

4.1.2. Scope of This Thesis

This work develops inconsistency measures for operator-induced goal conflicts in Category 1 and selected aspects of Category 2, focusing on conflicts detectable through reachability-based analysis of propositional PDDL planning problems. Specifically, three measures are proposed: the Goal Mutex Profile for operator-induced mutex relationships where goals are individually achievable but never satisfiable in a common reachable state, the Goal Sequencing Conflict Profile for irreversible action-based exclusions where achieving certain goals permanently blocks others, and the Unreachability Profile for generic unreachability diagnosis applicable across Category 2 subtypes.

Category 2b, circular dependencies, is recognized as an important source of unsolvability but is excluded from formal measurement in this thesis. Circular dependencies cause unreachability symptoms that are captured by the generic unreachability measure. Providing specific diagnostics for cycles, however, would require graph-theoretic analysis beyond reachability computation. While such analysis could yield more targeted repair suggestions, such as identifying which dependency edge to break versus which producer to add, this thesis prioritizes breadth of coverage over depth of cause-specific diagnosis. For instance, if no operator produces proposition *has_key*, a missing-producer case, the targeted repair is to add such an operator. By contrast, if the operator *activate_badge* has precondition *inside* and the operator *enter_building* has precondition *active_badge*, forming a circular dependency, the targeted repair is either to break one dependency edge by modifying an operator's preconditions, or to initialize one proposition in the cycle as initially true. The generic unreachability measure identifies which propositions are blocked. Distinguishing why they are blocked, whether missing producer, cycle, or action-based exclusion, is recognized as valuable future work but is orthogonal to the current goal-level and dependency-level framework.

Category 3, resource constraints, is also excluded from formal measurement, for two reasons. First, from a knowledge representation perspective, numeric resource insufficiency often reduces to missing-producer problems in Category 2a. Second, numeric fluents introduce significant complexity through PDDL 2.1 extensions [18], requiring fundamentally different measurement approaches. This thesis focuses on propositional PDDL planning with the `:strips` and `:typing` requirements. Numeric extensions warrant separate treatment. Future work may develop specialized measures for cycles and numeric resources, building on the foundational framework established here.

An important practical distinction emerges between static and dynamic analysis approaches. Static analyses operate directly on the problem specification $\langle P, O, I, G \rangle$ without state space exploration. Missing producers in Category 2a are detected by verifying whether $\exists o \in O$ with $p \in \text{add}(o)$. Operator-induced mutex in Category 1 admits a static over-approximation from delete-effect interactions, though exact detection requires reasoning about reachable states. Dynamic analyses, conversely, reason about reachable states or execution paths. Circular dependencies in Category 2b require graph analysis of precondition chains, exact unreachability across Category 2 requires forward or backward reachability computation, and irreversible action-based exclusions in Category 2c require trajectory analysis. This distinction has implications for computational complexity. Measures grounded in static over-approximation are typically more efficiently computable than those requiring exact dynamic state-space reasoning, motivating the bounded reachability approach adopted by the proposed measures.

Example 7 (Multiple Structural Categories)**.** To illustrate how multiple structural categories can manifest in a single planning problem, consider a security facility scenario:

$$
\begin{aligned}
P &= \{\text{outside}, \text{inside}, \text{has\_badge}, \text{vault\_accessed}, \text{lights\_on}, \text{lights\_off}\} \\
O &= \{\text{enter\_facility} = \langle \{\text{outside}, \text{has\_badge}\}, \{\text{inside}\}, \{\text{outside}\} \rangle, \\
&\quad \text{access\_vault} = \langle \{\text{inside}\}, \{\text{vault\_accessed}\}, \{\} \rangle, \\
&\quad \text{turn\_on} = \langle \{\text{lights\_off}\}, \{\text{lights\_on}\}, \{\text{lights\_off}\} \rangle, \\
&\quad \text{turn\_off} = \langle \{\text{lights\_on}\}, \{\text{lights\_off}\}, \{\text{lights\_on}\} \rangle\} \\
I &= \{\text{outside}, \text{lights\_off}\} \\
G &= \{\text{vault\_accessed}, \text{outside}, \text{lights\_on}, \text{lights\_off}\}
\end{aligned}
$$

This problem exhibits three distinct structural sources of unsolvability. First is Category 1, operator-induced mutex. The goal requires both *lights_on* and *lights_off*, but these propositions are mutually exclusive through operator effects, as achieving one necessarily deletes the other. Second is Category 2a, missing producer. The proposition *has_badge* appears in the precondition of *enter_facility* but no operator in O has *has_badge* in its add effects, making facility entry impossible regardless of other constraints. Third is Category 2c, action-based exclusion. Even if a badge-producing operator existed, *enter_facility* deletes *outside* without any operator to restore it, while the goal requires both *vault_accessed*, achievable only from inside, and *outside*, creating an irreversible conflict. The three categories illustrate structurally distinct measurement targets within a single problem. Category 1 and Category 2a each independently render the problem unsolvable, while Category 2c is present as a structural pattern that would persist even if a producer for *has_badge* were added. This demonstrates that the taxonomy identifies measurement targets rather than only top-level causes of unsolvability.

4.2. Motivation

A natural approach to measuring inconsistency in planning problems would encode the problem as a propositional knowledge base and apply classical inconsistency measures. This section demonstrates why such encodings yield uninformative results, motivating planning-native measures.

4.2.1. Encoding Planning Problems as Knowledge Bases

A propositional encoding of a planning problem $\langle P, O, I, G \rangle$ includes:

1. **Goals**: g for each $g \in G$
2. **Precondition dependencies**: For each $o \in O$ and each $p \in \text{add}(o)$, since achieving an add effect requires its preconditions:

$$p \rightarrow \bigwedge_{q \in \text{precond}(o)} q$$

3. **Closed-world assumptions**: $\neg p$ for propositions p that are neither in I nor produced by any operator

Example: Unreachability Encoding. Consider the Locked Door scenario (Scenario 1) where an agent must enter a room, but entry requires unlocking a door, which requires a key that no operator produces. The encoding yields:

$$\begin{aligned} K_{\text{LD}} = \{&\text{in\_room}, && \text{(goal)} \\ &\text{in\_room} \rightarrow \text{at\_door} \wedge \text{door\_unlocked}, && \text{(enter preconditions)} \\ &\text{door\_unlocked} \rightarrow \text{at\_door} \wedge \text{has\_key}, && \text{(unlock preconditions)} \\ &\neg\text{has\_key}\} && \text{(closed world, no producer)} \end{aligned}$$

This knowledge base is classically inconsistent. The goal *in_room* requires *door_unlocked* via modus ponens, which requires *has_key*, but $\neg$*has_key* is asserted.

4.2.2. Contension Measure Analysis

The contension measure $\mathcal{I}_c$ counts the minimum number of atoms assigned the conflicting value B in Priest's three-valued logic of paradox [35] to satisfy all formulas. The value B represents "both true and false." A formula is satisfied if it evaluates to T or B rather than F.

Proposition 1. $\mathcal{I}_c(K_{LD}) = 1$.

Proof. Consider the three-valued interpretation v defined by

$$v(\text{has\_key}) = B, \qquad v(\text{at\_door}) = v(\text{door\_unlocked}) = v(\text{in\_room}) = T.$$

Each formula of K_{LD} then evaluates to T or B:

$$\begin{aligned} \text{in\_room} &= T, \\ \text{in\_room} \to \text{at\_door} \wedge \text{door\_unlocked} &= T \to T \wedge T = T, \\ \text{door\_unlocked} \to \text{at\_door} \wedge \text{has\_key} &= T \to T \wedge B = B, \\ \neg\text{has\_key} &= \neg B = B. \end{aligned}$$

Hence v is a model of K_{LD}. Only *has_key* is assigned B, so $\mathcal{I}_c(K_{\text{LD}}) \leq 1$. Since K_{LD} is classically inconsistent, $\mathcal{I}_c(K_{\text{LD}}) \geq 1$. Therefore $\mathcal{I}_c(K_{\text{LD}}) = 1$. □

4.2.3. Limitations for Operator-Induced Conflicts

The Locked Door encoding detects inconsistency because the closed-world assumption ¬*has_key* follows directly from the STRIPS specification. No operator produces *has_key*, and it is not in I. For scenarios involving operator-induced conflicts, the encoding faces a more fundamental limitation that persists even when the encoding is enriched.

Example: Precondition-Only Encoding. Consider the Light Switch scenario (Scenario 3), where the goal requires both *on* and *off*, but toggle operators enforce mutual exclusion. Applying the encoding rules yields:

$$\begin{aligned} K_{\text{LS}} = \{&\text{on}, \text{off}, && \text{(goals)} \\ &\text{on} \to \text{off}, && \text{(turn\_on precondition)} \\ &\text{off} \to \text{on}\} && \text{(turn\_off precondition)} \end{aligned}$$

No closed-world negations apply since both propositions are producible. This knowledge base is classically consistent. The interpretation $v(\textit{on}) = v(\textit{off}) = T$ satisfies all formulas. The precondition-only encoding omits delete effects, so the mutual exclusion between *on* and *off* is not represented despite the planning problem being unsolvable.

Example: Enriched Encoding with Delete Effects. A natural response is to extend the encoding with delete-effect implications. For each operator o and each pair of an add effect p and a delete effect d of o, the additional rule $p \to \neg d$ captures that achieving p via o also removes d. For the Light Switch, this yields:

$$\begin{aligned} K^+_{\text{LS}} = \{&\text{on}, \text{off}, && \text{(goals)} \\ &\text{on} \to \text{off}, && \text{(turn\_on precondition)} \\ &\text{off} \to \text{on}, && \text{(turn\_off precondition)} \\ &\text{on} \to \neg\text{off}, && \text{(turn\_on deletes off)} \\ &\text{off} \to \neg\text{on}\} && \text{(turn\_off deletes on)} \end{aligned}$$

The enriched encoding is classically inconsistent. The goal *on* together with the deletion rule yields $\neg$*off*, contradicting the goal *off*. Yet $\mathcal{I}_c(K^+_{\text{LS}}) = 1$. Setting $v(\textit{off}) = B$ and $v(\textit{on}) = T$ satisfies every formula, since implications involving a B-valued atom evaluate to B rather than F.

Example: Sequencing Conflicts. The same pattern applies to the Trust and Travel scenario (Scenario 5), where an agent must secure a partnership requiring trust and exploit a remote opportunity, but traveling destroys trust irreversibly. The precondition-only encoding

$$\begin{aligned} K_{\text{TT}} = \{&\text{partnership}, \text{opportunity}, && \text{(goals)} \\ &\text{partnership} \to \text{at\_local} \wedge \text{trust}, && \text{(secure preconditions)} \\ &\text{opportunity} \to \text{at\_remote}, && \text{(exploit precondition)} \\ &\text{at\_remote} \to \text{at\_local}, && \text{(travel precondition)} \\ &\text{at\_local} \to \text{at\_remote}\} && \text{(return precondition)} \end{aligned}$$

is classically consistent. Setting all propositions to true satisfies every implication. Adding delete-effect implications for *travel* (at_remote $\to$ $\neg$at_local $\wedge$ $\neg$trust) and *return* (at_local $\to$ $\neg$at_remote) makes the encoding classically inconsistent. The goal chain *opportunity* $\to$ *at_remote* combined with both at_remote $\to$ at_local and at_remote $\to$ $\neg$at_local creates a contradiction whenever *at_remote* is true. Yet $\mathcal{I}_c = 1$. Assigning $v(\textit{at\_remote}) = B$ with all other propositions true satisfies every formula.

Beyond the uninformativeness of $\mathcal{I}_c$, the delete-effect implications are themselves an overapproximation. The rule $p \to \neg d$ fires whenever p is true, regardless of whether p was achieved by operator o or was already present in the initial state I. When multiple operators produce the same proposition with different delete effects, the encoding conflates their consequences. A more precise encoding would require tracking which operator produced each proposition, essentially reconstructing the state transition semantics that planning formalisms are designed to manage. Furthermore, sequencing conflicts are inherently path-dependent. The statement "achieving *opportunity* permanently blocks *partnership*" requires reasoning about trajectories through the state space. Propositional logic, being static, cannot express that from states where p holds, q becomes unreachable.

4.2.4. Why Direct Encoding Fails

The analysis reveals a consistent pattern. Across unreachability, mutex, and sequencing conflicts, $\mathcal{I}_c = 1$ regardless of problem structure. In each case, the three-valued B absorbs the conflict at a single proposition, satisfying all downstream implications. The contension measure therefore assigns identical severity to structurally distinct planning problems: a single missing producer (Locked Door), a pair of mutually exclusive goals (Light Switch), and an irreversible sequencing conflict (Trust and Travel) all yield $\mathcal{I}_c = 1$.

This uninformativeness stems from a mismatch between the measure and the domain. The contension measure is designed for static knowledge bases where each additional conflicting proposition reliably indicates greater inconsistency. Planning problems, however, exhibit structured conflicts with internal organization: dependency graphs of varying depth for unreachability, sets of mutex pairs for mutual exclusion, and directed blocking relationships for sequencing. These structures are invisible to any measure that operates on a flat propositional encoding.

These limitations hold regardless of encoding completeness. The precondition-only encoding fails to detect certain operator-induced conflicts entirely, while the enriched encoding with delete effects detects them but produces the same uninformative $\mathcal{I}_c = 1$ value. In neither case does the propositional approach provide actionable diagnostic information about the nature or severity of unsolvability.

These limitations motivate planning-native measures that operate directly on planning structures such as goals, operators, and reachable states, rather than reducing to propositional logic. The measures proposed in this section follow the inconsistency measurement methodology of formal definitions, rationality postulates, and complexity analysis, while respecting planning-specific semantics.

4.2.5. Design Principles

The adaptation strategy is guided by several key principles. Rather than reducing inconsistency to a single numerical severity score, the proposed measures provide diagnostic profiles consisting of multiple complementary values (formally defined in Definition 16). This recognizes that different aspects of unsolvability serve different debugging purposes, such as the number of affected goals versus the extent of broken dependencies.

A fundamental principle is to "measure what *is*, not what *should be*." Traditional repair-based inconsistency measures implicitly encode assumptions about valid repairs by counting minimal changes needed to restore consistency. In planning contexts, however, determining which components "should" exist requires domain expertise or exhaustive enumeration that is computationally prohibitive. Whether the fix involves missing operators, initial state facts, or goal modifications depends on the application. The proposed measures therefore report objective structural properties without presupposing which repairs are appropriate.

The structural taxonomy in Section 4.1 organizes unsolvability causes by their mathematical properties: operator-induced goal conflicts in Category 1, dependency and effect-structure failures in Category 2, and resource insufficiency in Category 3. This section focuses on Category 1 and Category 2 problems with three measures that exhibit formal relationships between them. The first two measures, Unreachability and Goal Mutex, perform state space analysis. They compute reachability once and analyze properties of the resulting state set. The third measure, Goal Sequencing Conflict (Sequencing for short), performs trajectory analysis. It examines path-dependent properties from goal-achieving states. Every sequencing conflict (g_1, g_2) with g_2 achievable implies that $\{g_1, g_2\}$ is a mutex goal pair, since the blocked goal cannot coexist with the blocking goal in any reachable state (formalized in Proposition 15). The measures therefore form a containment structure rather than detecting independent phenomena.

Each profile provides two complementary values following a standardized pattern: the scope variant reports goal-level severity, while the structure variant reveals underlying structural detail. Scope values are directly comparable across measures. Structure values provide diagnostic depth within each measure. Specifically:

- **Unreachability**: Structure expands from goal propositions to all propositions in the dependency graph, revealing cascading failure depth
- **Mutex**: Structure counts pairwise conflict relationships among affected goals, revealing conflict density
- **Sequencing**: Structure counts directed conflict pairs, revealing asymmetry in irreversible exclusions

4.3. Test Scenarios

This section introduces six test scenarios that serve as running examples throughout the theoretical development. Each scenario represents a small, focused planning problem exhibiting a specific structural pattern of unsolvability. The scenarios are organized by the measure designed to detect their failure mode, though some scenarios trigger multiple measures.

All scenarios are expressed as STRIPS planning problems $\langle P, O, I, G \rangle$ following the notation established in Section 2.2. For each scenario, the intuitive description, formal specification, and structural pattern are provided, along with forward reachability S_{reach}, formally defined in Definition 17. All test scenarios have small state spaces with $|P| \leq 8$ and converge within at most four steps ($h^* \leq 4$), as verified by direct inspection of the reachable state sets presented with each scenario. The examples therefore use the converged measures $\mathcal{I}(\Pi)$ throughout.

4.3.1. Unreachability Scenarios

The following scenarios exhibit dependency structure failures where required propositions never appear in any reachable state. They differ in dependency depth (chain length) versus width (parallel chains).

Scenario 1: Locked Door. An agent must enter a room, but entry requires unlocking a door, which requires a key that no operator produces. The dependency structure forms a linear chain: *in_room* depends on *door_unlocked*, which depends on *has_key*.

$$
\begin{aligned}
P &= \{\text{at\_door}, \text{has\_key}, \text{door\_unlocked}, \text{in\_room}\} \\
O &= \{\text{unlock} = \langle\{\text{at\_door}, \text{has\_key}\}, \{\text{door\_unlocked}\}, \{\}\rangle, \\
&\quad \text{enter} = \langle\{\text{at\_door}, \text{door\_unlocked}\}, \{\text{in\_room}\}, \{\text{at\_door}\}\rangle\} \\
I &= \{\text{at\_door}\} \\
G &= \{\text{in\_room}\}
\end{aligned}
$$

The *unlock* operator requires *has_key*, which has no producer and is not in I, blocking the entire dependency structure. Forward reachability yields $S_{\text{reach}} = \{\{\text{at\_door}\}\}$.

Scenario 2: Bank Vault. A more complex variant requires three independent precondition chains (two keys and admin approval) to achieve a single goal. Three independent dependency failures occur in parallel, each blocking a different precondition of the final operator.

$$
\begin{aligned}
P &= \{\text{at\_fac}, \text{key1}, \text{key2}, \text{admin\_appr}, \\
&\quad \text{d1\_open}, \text{d2\_open}, \text{acc\_granted}, \text{in\_vault}\} \\
O &= \{\text{open\_d1} = \langle\{\text{at\_fac}, \text{key1}\}, \{\text{d1\_open}\}, \{\}\rangle, \\
&\quad \text{open\_d2} = \langle\{\text{at\_fac}, \text{key2}\}, \{\text{d2\_open}\}, \{\}\rangle, \\
&\quad \text{grant\_acc} = \langle\{\text{admin\_appr}\}, \{\text{acc\_granted}\}, \{\}\rangle, \\
&\quad \text{enter} = \langle\{\text{at\_fac}, \text{d1\_open}, \text{d2\_open}, \text{acc\_granted}\}, \{\text{in\_vault}\}, \{\text{at\_fac}\}\rangle\} \\
I &= \{\text{at\_fac}\} \\
G &= \{\text{in\_vault}\}
\end{aligned}
$$

Three independent chains fail at their roots (*key1*, *key2*, *admin_appr* are all missing producers), blocking all downstream propositions. Forward reachability yields $S_{\text{reach}} = \{\{\text{at\_fac}\}\}$.

4.3.2. Mutex Scenarios

The following scenarios exhibit operator-induced mutual exclusions where goal propositions are each individually achievable but never true in any common reachable state. Actions are reversible. The system can transition between goal-satisfying states, but effects structurally prevent co-occurrence.

Scenario 3: Light Switch. A goal requires both *on* and *off* to be true, but toggle actions enforce mutual exclusion.

$$\begin{aligned} P &= \{\text{on}, \text{off}\} \\ O &= \{\text{turn\_on} = \langle\{\text{off}\}, \{\text{on}\}, \{\text{off}\}\rangle, \\ &\quad \text{turn\_off} = \langle\{\text{on}\}, \{\text{off}\}, \{\text{on}\}\rangle\} \\ I &= \{\text{off}\} \\ G &= \{\text{on}, \text{off}\} \end{aligned}$$

Forward reachability yields $S_{\text{reach}} = \{\{\text{off}\}, \{\text{on}\}\}$. Both goals appear in reachable states, but no state contains both. Actions are reversible (can toggle indefinitely), making this a pure mutex without sequencing conflict.

Scenario 4: Traffic Light. A goal requires all three colors (*red*, *yellow*, *green*), but cyclic transitions enforce that exactly one color is active at any time.

$$\begin{aligned} P &= \{\text{red}, \text{yellow}, \text{green}\} \\ O &= \{\text{red\_to\_green} = \langle\{\text{red}\}, \{\text{green}\}, \{\text{red}\}\rangle, \\ &\quad \text{green\_to\_yellow} = \langle\{\text{green}\}, \{\text{yellow}\}, \{\text{green}\}\rangle, \\ &\quad \text{yellow\_to\_red} = \langle\{\text{yellow}\}, \{\text{red}\}, \{\text{yellow}\}\rangle\} \\ I &= \{\text{red}\} \\ G &= \{\text{red}, \text{yellow}, \text{green}\} \end{aligned}$$

Forward reachability yields $S_{\text{reach}} = \{\{\text{red}\}, \{\text{green}\}, \{\text{yellow}\}\}$. All three goals are individually achievable (the cycle is complete), but no state contains more than one. Every pair of goals is mutually exclusive, forming a complete conflict graph where all $\binom{3}{2} = 3$ pairs are in conflict.

4.3.3. Sequencing Conflict Scenarios

The following scenarios exhibit irreversible exclusions where achieving certain goal propositions creates permanent dead-end states. Unlike mutex scenarios where goals alternate via reversible actions, sequencing conflicts involve one-way transitions. Consistent with the measure relationships (Proposition 15), these scenarios exhibit both mutex and sequencing conflict values.

Scenario 5: Trust and Travel. An agent must secure a local partnership that requires established trust, and exploit a remote opportunity. Traveling destroys the trust relationship irreversibly. This creates a unidirectional sequencing conflict.

$$
\begin{aligned}
P &= \{\text{at\_local}, \text{at\_remote}, \text{trust}, \text{partnership}, \text{opportunity}\} \\
O &= \{\text{secure} = \langle\{\text{at\_local}, \text{trust}\}, \{\text{partnership}\}, \{\}\rangle, \\
&\quad \text{travel} = \langle\{\text{at\_local}\}, \{\text{at\_remote}\}, \{\text{at\_local}, \text{trust}, \text{partnership}\}\rangle, \\
&\quad \text{exploit} = \langle\{\text{at\_remote}\}, \{\text{opportunity}\}, \{\}\rangle, \\
&\quad \text{return} = \langle\{\text{at\_remote}\}, \{\text{at\_local}\}, \{\text{at\_remote}\}\rangle\} \\
I &= \{\text{at\_local}, \text{trust}\} \\
G &= \{\text{partnership}, \text{opportunity}\}
\end{aligned}
$$

Forward reachability yields

$$
\begin{aligned}
S_{\text{reach}} = \{&\{\text{at\_local}, \text{trust}\}, \{\text{at\_local}, \text{trust}, \text{partnership}\}, \{\text{at\_remote}\}, \\
&\{\text{at\_remote}, \text{opportunity}\}, \{\text{at\_local}\}, \{\text{at\_local}, \text{opportunity}\}\}.
\end{aligned}
$$

Both goals are individually achievable: *partnership* via *secure*, and *opportunity* via *travel* $\rightarrow$ *exploit*. However, the goals never coexist (mutex). The conflict is asymmetric:

- $(\text{opportunity}, \text{partnership})$ **is** a sequencing conflict: from any state with *opportunity*, the agent lacks *trust* to achieve *partnership* (travel destroyed it).
- $(\text{partnership}, \text{opportunity})$ is **not** a sequencing conflict: from a *partnership*-state, *opportunity* remains reachable via *travel* $\rightarrow$ *exploit* (though this path deletes *partnership*).

Scenario 6: Rival Alliances. An agent seeks alliances with two rival factions, but each faction demands exclusive loyalty. Allying with one faction alienates the other. This creates a symmetric sequencing conflict.

$$
\begin{aligned}
P &= \{\text{neutral\_A}, \text{neutral\_B}, \text{allied\_A}, \text{allied\_B}\} \\
O &= \{\text{ally\_A} = \langle\{\text{neutral\_A}\}, \{\text{allied\_A}\}, \{\text{neutral\_B}\}\rangle, \\
&\quad \text{ally\_B} = \langle\{\text{neutral\_B}\}, \{\text{allied\_B}\}, \{\text{neutral\_A}\}\rangle\} \\
I &= \{\text{neutral\_A}, \text{neutral\_B}\} \\
G &= \{\text{allied\_A}, \text{allied\_B}\}
\end{aligned}
$$

Forward reachability yields:

$$
S_{\text{reach}} = \{\{\text{neutral\_A}, \text{neutral\_B}\}, \{\text{neutral\_A}, \text{allied\_A}\}, \{\text{neutral\_B}, \text{allied\_B}\}\}
$$

Both goals are individually achievable but never coexist (mutex). Unlike Trust and Travel, both directions are sequencing conflicts. Allying with either faction alienates the other, deleting the precondition for the remaining alliance.

4.3.4. Scenario Summary

Table 1 summarizes the test scenarios and their structural characteristics.

Scenario	Category	Pattern	Measure
Locked Door	2a	Linear chain	$\mathcal{I}_{\text{UR}}$
Bank Vault	2a	Parallel chains	$\mathcal{I}_{\text{UR}}$
Light Switch	1	Simple pair	$\mathcal{I}_{\text{MX}}$
Traffic Light	1	Complete clique	$\mathcal{I}_{\text{MX}}$
Trust and Travel	1, 2c	Unidirectional	$\mathcal{I}_{\text{GS}}$
Rival Alliances	1, 2c	Symmetric	$\mathcal{I}_{\text{GS}}$

Table 1: Test scenarios and their structural patterns.

The structural patterns referenced in Table 1 are characterized as follows:

- **Linear chain**: a single dependency path from the goal to an unproducible root.
- **Parallel chains**: multiple independent dependency paths to one goal operator.
- **Simple pair**: exactly one mutex pair among the goal propositions.
- **Complete clique**: every pair of goal propositions is mutually exclusive.
- **Unidirectional**: one goal permanently blocks the other, not vice versa.
- **Symmetric**: achieving either goal permanently blocks the other.

These scenarios reappear in Section 4.4 to illustrate measure definitions, in Section 4.5.1 to demonstrate postulate compliance, and in Table 6 as concrete measure values for comparison.

4.4. Inconsistency Measures for Planning Problems

This section presents the three diagnostic measures with formal definitions. Each measure is defined, followed by examples using the test scenarios.

4.4.1. Preliminary Definitions

Before defining the measures themselves, several foundational concepts are established.

Definition 16 (Diagnostic Profile)**.** A *diagnostic profile* is a pair of non-negative integers $(\mathcal{I}^{\text{scope}}, \mathcal{I}^{\text{struct}})$. The scope component $\mathcal{I}^{\text{scope}}$ counts goal propositions exhibiting a structural phenomenon, while the structure component $\mathcal{I}^{\text{struct}}$ counts the underlying structural relationships.

Definition 17 (Reachable States)**.** Let $\langle P, O, I, G\rangle$ be a planning problem. The set of *reachable states* $S_{\text{reach}} \subseteq 2^P$ is the intersection of all sets of states satisfying:

1. $I \in S_{\text{reach}}$
2. If $s \in S_{\text{reach}}$ and $o \in O$ with $\text{precond}(o) \subseteq s$, then $\text{RESULT}(s, o) \in S_{\text{reach}}$

The set S_{reach} can be computed via forward reachability analysis. Starting from I, iteratively apply all applicable operators until no new states are generated. Since P is finite and each state is a subset of P, this process terminates in at most $2^{|P|}$ iterations. Empirical convergence behavior on benchmark problems is discussed in Section 6.

Definition 18 (Forward Reachability from State)**.** Let $\langle P, O, I, G\rangle$ be a planning problem. For any state $s \subseteq P$, the set of *forward-reachable states from* s, denoted $\text{REACH}(s)$, is the intersection of all sets of states satisfying:

1. $s \in \text{REACH}(s)$
2. If $s' \in \text{REACH}(s)$ and $o \in O$ with $\text{precond}(o) \subseteq s'$, then $\text{RESULT}(s', o) \in \text{REACH}(s)$

Note that $S_{\text{reach}} = \text{REACH}(I)$, and for any $s \in S_{\text{reach}}$, $\text{REACH}(s) \subseteq S_{\text{reach}}$.

4.4.2. Horizon-Bounded Reachability

The set S_{reach} as defined above is the complete set of reachable states under unbounded exploration. In practice, computing S_{reach} requires enumerating the full reachable state space, which can contain up to $2^{|P|}$ states. For computational purposes, the exploration depth is bounded by a horizon parameter $h \in \mathbb{N}$.

Definition 19 (Horizon-Bounded Reachable States)**.** Let $\Pi = \langle P, O, I, G\rangle$ be a planning problem and $h \in \mathbb{N}$. The set of *horizon-bounded reachable states* $S^h_{\text{reach}} \subseteq 2^P$ is defined by the step-indexed construction:

$$S^0_{\text{reach}} = \{I\}$$
$$S^{i+1}_{\text{reach}} = S^i_{\text{reach}} \cup \{\text{RESULT}(s, o) \mid s \in S^i_{\text{reach}},\ o \in O,\ \text{precond}(o) \subseteq s\}\,.$$

Equivalently, S^h_{reach} is the set of all states reachable from I by applying at most h operators in sequence.

This parameter plays an analogous role to the trace length parameter m in linear temporal logic on fixed traces (LTL_{ff}), a temporal formalism in which formulas are evaluated over traces of a fixed length m rather than infinite models, so m enters the semantics as an explicit parameter. In that setting, Corea et al. [8] define inconsistency measures as functions $\mathcal{I}\colon \mathbb{K} \times \mathbb{N}_0 \to \mathbb{R}^\infty_{\geq 0}$ where $\mathbb{K}$ denotes the set of LTL_{ff} knowledge bases and the second argument is the trace length, and Kuhlmann [28] follows the same convention. The measures defined below adopt this approach and take the planning problem and the horizon as input.

The following properties relate bounded and unbounded reachability.

Proposition 2 (Monotonicity and Convergence of Bounded Reachability)**.** *Let* Π *be a planning problem. Then:*

1. (Monotonicity) $S^h_{reach} \subseteq S^{h+1}_{reach}$ *for all* $h \in \mathbb{N}$.
2. (Convergence) *There exists a finite* $h^* \leq 2^{|P|}$ *such that* $S^h_{reach} = S_{reach}$ *for all* $h \geq h^*$.

Proof. Monotonicity. By construction, $S^{i+1}_{\text{reach}} = S^i_{\text{reach}} \cup X$ for some set X, so $S^i_{\text{reach}} \subseteq S^{i+1}_{\text{reach}}$ for every $i \in \mathbb{N}$. In particular, $S^h_{\text{reach}} \subseteq S^{h+1}_{\text{reach}}$.

Convergence. The sequence $(S^i_{\text{reach}})_{i\in\mathbb{N}}$ is a non-decreasing family of subsets of the finite set 2^P, which contains at most $2^{|P|}$ states. A strictly increasing chain can grow at most $2^{|P|}$ times before it saturates, so there exists $h^* \leq 2^{|P|}$ with $S^{h^*+1}_{\text{reach}} = S^{h^*}_{\text{reach}}$. Once the sequence stabilizes, the recurrence coincides with the inductive definition of $\text{REACH}(I)$, so $S^{h^*}_{\text{reach}} = \text{REACH}(I) = S_{\text{reach}}$. By monotonicity, $S^h_{\text{reach}} = S_{\text{reach}}$ for all $h \geq h^*$. □

In LTL_{ff}, a sufficient trace length is computable from the syntactic depth of the formulas. Corea et al. [8] show that setting $m \geq d(\mathcal{K})$ guarantees that a three-valued interpretation exists. No analogous efficient bound exists for h^* in the planning setting. Computing the state space diameter, which provides an upper bound on h^*, is PSPACE-hard for factored transition systems, as shown by Abdulaziz et al. [1]. This intractability motivates the explicit parameterization of the measures by h. Since no efficiently computable sufficient horizon is available in general, the dependence on h is made part of the formal measure definition rather than treated as a purely implementational concern.

An important consequence of the horizon bound is that the mutex and sequencing measures exhibit non-monotonic behavior in h. Increasing h both makes new goal propositions reachable, which may introduce apparent conflicts among newly achievable goals, and provides new execution traces, which may resolve previously reported conflicts through longer witness paths. This contrasts with the unreachability measure, where $\mathcal{I}^{\text{scope}}_{\text{UR}}$ is monotonically non-increasing in h as a direct consequence of the monotonicity of S^h_{reach}. The non-monotonic behavior of the mutex and sequencing measures is demonstrated empirically in Section 6.4.

4.4.3. Unreachability Profile

The Unreachability Profile provides a unified approach to Category 2 unsolvability by characterizing unreachable propositions through reachability analysis. This profile applies uniformly to all dependency structure subtypes: missing producers (2a), circular dependencies (2b), and dead-end states (2c).

The profile employs forward reachability analysis to compute: (1) the number of goal propositions not appearing in any reachable state, and (2) the total number of propositions not appearing in any reachable state. The first value identifies which

goal propositions are blocked, while the second reveals the scope of broken dependencies. The difference between these values indicates cascading unreachability depth.

Definition 20 (Unreachable Goal Propositions Count)**.** Let $\Pi = \langle P, O, I, G \rangle$ be a planning problem and $h \in \mathbb{N}$, with horizon-bounded reachable states S^h_{reach}. The measure $\mathcal{I}^{\text{scope}}_{\text{UR}}$ counting goal propositions that are unreachable is defined as:

$$\mathcal{I}^{\text{scope}}_{\text{UR}}(\Pi, h) = |\{g \in G \mid \nexists s \in S^h_{\text{reach}} : g \in s\}|$$

This measure indicates the scope of unreachability problems at the goal level. It counts how many goal propositions cannot be achieved. The value ranges from 0 (all goals reachable) to $|G|$ (no goals reachable).

Definition 21 (Total Unreachable Propositions Count)**.** Let $\Pi = \langle P, O, I, G \rangle$ be a planning problem and $h \in \mathbb{N}$, with horizon-bounded reachable states S^h_{reach}. The measure $\mathcal{I}^{\text{struct}}_{\text{UR}}$ counting all unreachable propositions is defined as:

$$\mathcal{I}^{\text{struct}}_{\text{UR}}(\Pi, h) = |\{p \in P \mid \nexists s \in S^h_{\text{reach}} : p \in s\}|$$

This measure reveals the structure of dependency problems by counting all unreachable propositions throughout the proposition space. The value ranges from 0 to $|P|$. The difference $\mathcal{I}^{\text{struct}}_{\text{UR}} - \mathcal{I}^{\text{scope}}_{\text{UR}}$ indicates the depth of cascading dependencies. Together, $\mathcal{I}^{\text{scope}}_{\text{UR}}$ and $\mathcal{I}^{\text{struct}}_{\text{UR}}$ form the Unreachability Profile.

Note that $\mathcal{I}^{\text{struct}}_{\text{UR}}$ counts all unreachable propositions including those not on any dependency path to a goal. Propositions that are unreachable but irrelevant to goal achievement contribute to the count. This provides a conservative upper bound on dependency-relevant unreachability; filtering to goal-relevant propositions is left as a possible refinement.

As established in Section 4.3, the test scenarios use converged measures $\mathcal{I}(\Pi)$.

Example: Locked Door (Scenario 1). For the Locked Door scenario, forward reachability analysis yields $S_{\text{reach}} = \{\{\text{at\_door}\}\}$ since *unlock* requires *has_key*, which has no producer. The Unreachability Profile yields:

$$\mathcal{I}^{\text{scope}}_{\text{UR}}(\Pi) = |\{\text{in\_room}\}| = 1$$
$$\mathcal{I}^{\text{struct}}_{\text{UR}}(\Pi) = |\{\text{has\_key}, \text{door\_unlocked}, \text{in\_room}\}| = 3$$

The difference of 2 indicates two intermediate propositions in the dependency structure that are unreachable due to the missing *has_key*. Note that *at_door* is reachable (it is in I) and thus not counted.

Example: Bank Vault (Scenario 2). For the Bank Vault scenario, reachability analysis yields $S_{\text{reach}} = \{\{\text{at\_fac}\}\}$ since none of the root preconditions (*key1*, *key2*, *admin_appr*) are initially true or produced by any operator. The Unreachability Profile yields:

$$\begin{aligned}\mathcal{I}_{\text{UR}}^{\text{scope}}(\Pi) &= |\{\text{in\_vault}\}| = 1\\ \mathcal{I}_{\text{UR}}^{\text{struct}}(\Pi) &= |\{\text{key1}, \text{key2}, \text{admin\_appr},\\ &\qquad \text{d1\_open}, \text{d2\_open}, \text{acc\_granted},\\ &\qquad \text{in\_vault}\}| = 7\end{aligned}$$

This demonstrates how $\mathcal{I}_{\text{UR}}^{\text{struct}}$ distinguishes shallow dependency problems (difference of 2 in Locked Door) from deep cascading failures (difference of 6 in Bank Vault), revealing three independent missing precondition chains converging on a single goal. A small difference between scope and structure indicates shallow failures; a large difference reveals deep cascading dependencies.

4.4.4. Goal Mutex Profile

The Goal Mutex Profile addresses unsolvability where goal propositions are each individually achievable but never true in any common reachable state. This differs from specification-level contradictions (goals containing explicit negation like $\{p, \neg p\}$, which do not exist in propositional STRIPS) and from sequencing conflicts (where achieving one goal permanently blocks another). Mutex relationships are operator-induced. Actions enforce mutual exclusion through their effects, creating states where goals alternate but never coexist.

Definition 22 (Mutex Goal Pair)**.** Let $\Pi = \langle P, O, I, G\rangle$ be a planning problem and $h \in \mathbb{N}$, with horizon-bounded reachable states S_{reach}^{h}. Goal propositions $g_1, g_2 \in G$ with $g_1 \neq g_2$ are *mutex at horizon* h if:

1. $\exists s_1 \in S_{\text{reach}}^{h} : g_1 \in s_1$ (goal g_1 is achievable)
2. $\exists s_2 \in S_{\text{reach}}^{h} : g_2 \in s_2$ (goal g_2 is achievable)
3. $\nexists s \in S_{\text{reach}}^{h} : \{g_1, g_2\} \subseteq s$ (goals never co-occur)

This definition captures operator-induced exclusions. Both goals are individually reachable, but action effects prevent them from holding in the same state. Note that mutex is a symmetric relation. If (g_1, g_2) is mutex, then (g_2, g_1) is also mutex.

Definition 23 (Mutex Goals Count)**.** Let $\Pi = \langle P, O, I, G\rangle$ be a planning problem and $h \in \mathbb{N}$. The measure $\mathcal{I}_{\text{MX}}^{\text{scope}}$ counting goal propositions involved in mutex relationships is defined as:

$$\mathcal{I}_{\text{MX}}^{\text{scope}}(\Pi, h) = |\{g \in G \mid \exists g' \in G, g \neq g' : \text{mutex}_h(g, g')\}|$$

This measure indicates the scope of mutex problems at the goal level. It counts how many goal propositions participate in at least one mutex relationship. The value ranges from 0 (no mutex) to $|G|$ (all goals involved in mutex).

Definition 24 (Mutex Pair Count)**.** Let $\Pi = \langle P, O, I, G \rangle$ be a planning problem and $h \in \mathbb{N}$. The measure $\mathcal{I}_{\text{MX}}^{\text{struct}}$ counting two-element subsets of G whose elements are mutex is defined as:

$$\mathcal{I}_{\text{MX}}^{\text{struct}}(\Pi, h) = |\{\{g_1, g_2\} \mid g_1, g_2 \in G, g_1 \neq g_2, \text{mutex}_h(g_1, g_2)\}|$$

This measure reveals the structure of mutex relationships by counting two-element subsets rather than ordered pairs, reflecting the symmetry of the mutex relation. The value ranges from 0 to $\binom{|G|}{2}$. The ratio between $\mathcal{I}_{\text{MX}}^{\text{scope}}$ and $\mathcal{I}_{\text{MX}}^{\text{struct}}$ reveals conflict patterns: a small ratio suggests isolated mutex pairs, while a large ratio (approaching $\binom{|G|}{2}$) indicates dense mutex cliques. Together, $\mathcal{I}_{\text{MX}}^{\text{scope}}$ and $\mathcal{I}_{\text{MX}}^{\text{struct}}$ form the Goal Mutex Profile.

Example: Light Switch (Scenario 3). For the Light Switch scenario, forward reachability analysis yields $S_{\text{reach}} = \{\{\text{off}\}, \{\text{on}\}\}$. Both goal propositions appear in reachable states, but no state contains both. The Goal Mutex Profile yields:

$$\mathcal{I}_{\text{MX}}^{\text{scope}}(\Pi) = |\{\text{on}, \text{off}\}| = 2$$
$$\mathcal{I}_{\text{MX}}^{\text{struct}}(\Pi) = |\{\{\text{on}, \text{off}\}\}| = 1$$

The profile detects operator-induced mutual exclusion despite reversibility. Actions allow transitions between states, but effects structurally prevent co-occurrence.

Example: Traffic Light (Scenario 4). For the Traffic Light scenario, reachability analysis yields $S_{\text{reach}} = \{\{\text{red}\}, \{\text{green}\}, \{\text{yellow}\}\}$. All three goal propositions are individually achievable (the system is cyclic), but no state contains more than one color. The Goal Mutex Profile yields:

$$\mathcal{I}_{\text{MX}}^{\text{scope}}(\Pi) = |\{\text{red}, \text{yellow}, \text{green}\}| = 3$$
$$\mathcal{I}_{\text{MX}}^{\text{struct}}(\Pi) = |\{\{\text{red}, \text{yellow}\}, \{\text{red}, \text{green}\}, \{\text{yellow}, \text{green}\}\}| = 3$$

This demonstrates a complete mutex clique. All goal pairs are mutually exclusive, yielding $\mathcal{I}_{\text{MX}}^{\text{struct}}(\Pi) = \binom{3}{2} = 3$. The profile distinguishes this dense conflict structure from isolated mutex pairs.

4.4.5. Goal Sequencing Conflict Profile

The Goal Sequencing Conflict Profile addresses Category 2c unsolvability, where achieving certain goal propositions creates dead-end states that prevent other goal propositions from being reached. Unlike mutex relationships where goals can be achieved in different states but never in the same state, sequencing conflicts involve irreversible state transitions where achieving one goal permanently blocks another.

Definition 25 (Sequencing Conflict)**.** Let $\Pi = \langle P, O, I, G \rangle$ be a planning problem and $h \in \mathbb{N}$, with horizon-bounded reachable states S^h_{reach}. For goal propositions $g_1, g_2 \in G$ with $g_1 \neq g_2$, a *sequencing conflict at horizon* h, denoted (g_1, g_2), exists if:

1. $\exists s \in S^h_{\text{reach}} : g_1 \in s$ (goal g_1 is achievable)
2. $\exists s \in S^h_{\text{reach}} : g_2 \in s$ (goal g_2 is achievable)
3. There is no execution trace $\sigma_0, \sigma_1, \ldots, \sigma_k$ from I with $k \leq h$ such that $g_1 \in \sigma_i$ and $g_2 \in \sigma_j$ for some $0 \leq i \leq j \leq k$

Informally, (g_1, g_2) is a sequencing conflict at horizon h when both goals are individually achievable, but no single execution trace of length at most h achieves g_1 and subsequently g_2. Note that sequencing conflict is directional: (g_1, g_2) being a sequencing conflict does not imply (g_2, g_1) is. Each ordered pair is checked independently. The requirement that g_2 be achievable (condition 2) ensures the conflict is meaningful, distinguishing "g_1 blocks g_2" from "g_2 was never achievable anyway."

At sufficient horizon, condition 3 is equivalent to requiring that for every reachable state containing g_1, no state containing g_2 is forward-reachable from it. The definition thus captures permanent blocking at convergence, where achieving g_1 creates an irreversible dead-end from which g_2 can never be satisfied. This contrasts with mutex relationships (Definition 22), where goals may alternate between states via reversible actions but cannot co-occur.

Definition 26 (Conflicting Goal Propositions Count)**.** Let $\Pi = \langle P, O, I, G \rangle$ be a planning problem and $h \in \mathbb{N}$. The measure $\mathcal{I}^{\text{scope}}_{\text{GS}}$ counting goal propositions involved in sequencing conflicts is defined as:

$$\mathcal{I}^{\text{scope}}_{\text{GS}}(\Pi, h) = |\{g \in G \mid \exists g' \in G \text{ such that } (g, g') \text{ or } (g', g) \text{ is a sequencing conflict at horizon } h\}|$$

This measure indicates the scope of sequencing conflicts. It counts how many goal propositions participate in at least one action-based exclusion. The value ranges from 0 (no sequencing conflicts) to $|G|$ (all goals involved).

Definition 27 (Sequencing Conflict Pair Count)**.** Let $\Pi = \langle P, O, I, G \rangle$ be a planning problem and $h \in \mathbb{N}$. The measure $\mathcal{I}^{\text{struct}}_{\text{GS}}$ counting ordered sequencing conflict pairs is defined as:

$$\mathcal{I}^{\text{struct}}_{\text{GS}}(\Pi, h) = |\{(g_1, g_2) \mid g_1, g_2 \in G, g_1 \neq g_2, (g_1, g_2) \text{ is a sequencing conflict at horizon } h\}|$$

This measure reveals the structure of sequencing conflicts by counting ordered pairs. Unlike mutex pairs (which are unordered), sequencing conflicts have direction: (g_1, g_2) means achieving g_1 blocks g_2, but the reverse may not hold. The value ranges from 0 to $|G| \times (|G| - 1)$. Together, $\mathcal{I}^{\text{scope}}_{\text{GS}}$ and $\mathcal{I}^{\text{struct}}_{\text{GS}}$ form the Goal Sequencing Conflict Profile.

Example: Trust and Travel (Scenario 5). For the Trust and Travel scenario, forward reachability yields states including

$$\{\text{at\_local}, \text{trust}, \text{partnership}\} \quad \text{and} \quad \{\text{at\_local}, \text{opportunity}\},$$

but no state contains both goals.

Both goals are individually achievable but never coexist, satisfying the mutex definition. Additionally, $(\text{opportunity}, \text{partnership})$ is a sequencing conflict. From any state with *opportunity*, the agent lacks *trust* to achieve *partnership*. However, $(\text{partnership}, \text{opportunity})$ is not a sequencing conflict. From *partnership*-states, *opportunity* remains reachable via *travel* $\rightarrow$ *exploit* (though this path deletes *partnership*).

The Goal Mutex Profile yields:

$$\mathcal{I}_{\text{MX}}^{\text{scope}}(\Pi) = |\{\text{partnership}, \text{opportunity}\}| = 2$$
$$\mathcal{I}_{\text{MX}}^{\text{struct}}(\Pi) = |\{\{\text{partnership}, \text{opportunity}\}\}| = 1$$

The Goal Sequencing Conflict Profile yields:

$$\mathcal{I}_{\text{GS}}^{\text{scope}}(\Pi) = |\{\text{partnership}, \text{opportunity}\}| = 2$$
$$\mathcal{I}_{\text{GS}}^{\text{struct}}(\Pi) = |\{(\text{opportunity}, \text{partnership})\}| = 1$$

This demonstrates that irreversible problems exhibit both mutex and sequencing. The goals cannot coexist (mutex), and the sequencing measure additionally identifies the direction of the conflict.

Example: Rival Alliances (Scenario 6). For the Rival Alliances scenario, forward reachability yields

$$S_{\text{reach}} = \{\{\text{neutral\_A}, \text{neutral\_B}\}, \{\text{neutral\_A}, \text{allied\_A}\}, \{\text{neutral\_B}, \text{allied\_B}\}\}.$$

Both goals are individually achievable but never coexist, and allying with either faction alienates the other.

The Goal Mutex Profile yields:

$$\mathcal{I}_{\text{MX}}^{\text{scope}}(\Pi) = |\{\text{allied\_A}, \text{allied\_B}\}| = 2$$
$$\mathcal{I}_{\text{MX}}^{\text{struct}}(\Pi) = |\{\{\text{allied\_A}, \text{allied\_B}\}\}| = 1$$

The Goal Sequencing Conflict Profile yields:

$$\mathcal{I}_{\text{GS}}^{\text{scope}}(\Pi) = |\{\text{allied\_A}, \text{allied\_B}\}| = 2$$
$$\mathcal{I}_{\text{GS}}^{\text{struct}}(\Pi) = |\{(\text{allied\_A}, \text{allied\_B}), (\text{allied\_B}, \text{allied\_A})\}| = 2$$

This demonstrates symmetric exclusions where both alliances block each other. Like Trust and Travel, it exhibits both mutex and sequencing, but with $\mathcal{I}_{\text{GS}}^{\text{struct}}(\Pi) = 2$ (bidirectional) rather than 1 (unidirectional). The sequencing structure value thus distinguishes symmetric from asymmetric irreversibility.

This measure provides directional information beyond mutex. Since blocking implies non-coexistence, sequencing conflicts form a subset of mutex relationships, while the ordered pair count reveals which goal achievement causes the dead-end. The Light Switch scenario shows reversible mutex without sequencing, whereas Trust and Travel exhibits both.

4.4.6. Measure Range Summary

Table 2 summarizes the value ranges of all six measure components.

Measure	Range	Maximum
$\mathcal{I}_{\text{UR}}^{\text{scope}}$	$[0, \lvert G\rvert]$	All goals unreachable
$\mathcal{I}_{\text{UR}}^{\text{struct}}$	$[0, \lvert P\rvert]$	All propositions unreachable
$\mathcal{I}_{\text{MX}}^{\text{scope}}$	$[0, \lvert G\rvert]$	All goals in mutex relationships
$\mathcal{I}_{\text{MX}}^{\text{struct}}$	$[0, \binom{\lvert G\rvert}{2}]$	All goal pairs are mutex
$\mathcal{I}_{\text{GS}}^{\text{scope}}$	$[0, \lvert G\rvert]$	All goals in sequencing conflicts
$\mathcal{I}_{\text{GS}}^{\text{struct}}$	$[0, \lvert G\rvert(\lvert G\rvert - 1)]$	All ordered pairs are conflicts

Table 2: Value ranges of proposed inconsistency measures.

4.5. Theoretical Properties

This section analyzes the theoretical foundations of the proposed measures: rationality postulates adapted for planning problems, formal relationships between the measures, and computational complexity.

4.5.1. Rationality Postulates

A core component of inconsistency measurement research, as developed by Hunter and Konieczny [27] and surveyed by Thimm [38], is analyzing which rationality postulates measures satisfy. As demonstrated by Ulbricht et al. [40], domain-specific adaptations may require modified postulates. Classical postulates cannot be applied directly to planning problems for two reasons. First, planning has fundamentally different structure than propositional knowledge bases, involving goals, operators, and states rather than formulas. Second, planning exhibits inherently non-monotonic behavior, as adding operators can reduce inconsistency by enabling new solution paths. This thesis therefore introduces three planning-adapted postulates, namely Planning Consistency, Operator Monotonicity, and Goal Monotonicity, defined below.

Definition 28 (Planning Consistency)**.** An inconsistency measure $\mathcal{I}$ for planning problems satisfies *Planning Consistency* if for all Π and all sufficiently large h:

$$\mathcal{I}(\Pi, h) = 0 \text{ iff } \Pi \text{ is solvable}$$

This adapts the classical Consistency postulate ($\mathcal{I}(K) = 0$ iff K is consistent) to the planning domain, where solvability replaces logical consistency. The requirement of "sufficiently large h" ensures the horizon is at or beyond the convergence point h^* (Proposition 2), so that the measure reflects the true reachable state space rather than an approximation.

Definition 29 (Operator Monotonicity)**.** An inconsistency measure $\mathcal{I}$ satisfies *Operator Monotonicity* if for all planning problems $\Pi = \langle P, O, I, G \rangle$, operators o, and horizons h:

$$\mathcal{I}(\langle P, O \cup \{o\}, I, G \rangle, h) \leq \mathcal{I}(\Pi, h)$$

Note that Operator Monotonicity reverses classical monotonicity because adding operators can only enable new paths, potentially reducing inconsistency. This parallels the observation by Ulbricht et al. [40] that non-monotonic logics require inverted monotonicity postulates.

Definition 30 (Goal Monotonicity)**.** An inconsistency measure $\mathcal{I}$ satisfies *Goal Monotonicity* if for all planning problems $\Pi = \langle P, O, I, G \rangle$, goals g, and horizons h:

$$\mathcal{I}(\langle P, O, I, G \cup \{g\} \rangle, h) \geq \mathcal{I}(\Pi, h)$$

Goal Monotonicity preserves the direction of classical monotonicity. Adding requirements (goals) can only increase or maintain inconsistency, never decrease it. The following propositions analyze postulate compliance for each measure. Counterexamples use the converged test scenarios from Section 4.3, so the proofs write $\mathcal{I}(\Pi)$ for the converged measure values.

Proposition 3. *$\mathcal{I}_{UR}^{scope}$ does not satisfy Planning Consistency.*

Proof. Counterexample: In the Light Switch problem both goals *on* and *off* appear in reachable states at any $h \geq 1$, so $\mathcal{I}_{\text{UR}}^{\text{scope}}(\Pi) = 0$, yet the problem is unsolvable because the goals are mutex. This violates Planning Consistency, which requires $\mathcal{I} = 0$ only when Π is solvable. □

Proposition 4. *$\mathcal{I}_{UR}^{scope}$ satisfies Operator Monotonicity.*

Proof. Adding an operator o can only expand S^h_{reach} or leave it unchanged, since every execution trace available with O remains available with $O \cup \{o\}$. If goal g was unreachable (not appearing in any state in S^h_{reach}), adding o may make it reachable by enabling new state transitions. This can only reduce or maintain $\mathcal{I}_{\text{UR}}^{\text{scope}}$, never increase it. □

Proposition 5. *$\mathcal{I}_{UR}^{scope}$ satisfies Goal Monotonicity.*

Proof. Adding goal g to G results in exactly one of two cases: (a) g appears in some state in S^h_{reach}, so $\mathcal{I}_{\text{UR}}^{\text{scope}}$ is unchanged, or (b) g does not appear in any state in S^h_{reach}, so $\mathcal{I}_{\text{UR}}^{\text{scope}}$ increases by exactly 1. □

Proposition 6. *$\mathcal{I}_{MX}^{scope}$ does not satisfy Planning Consistency.*

Proof. Counterexample: In the Locked Door problem no mutex relationships exist because the goal *in_room* is not achievable, so $\mathcal{I}_{\text{MX}}^{\text{scope}}(\Pi) = 0$, yet the problem is unsolvable. □

Proposition 7. *$\mathcal{I}_{MX}^{scope}$ does not satisfy Operator Monotonicity.*

Proof. Counterexample: Consider $P = \{a, g_1, g_2\}$, $I = \{a\}$, $G = \{g_1, g_2\}$, and initial operator set $O = \{\text{reach\_g1}\}$ where reach_g1 $= \langle\{a\}, \{g_1\}, \{a\}\rangle$. Goal g_1 is achievable but g_2 is not, so no mutex exists and $\mathcal{I}_{\text{MX}}^{\text{scope}}(\Pi) = 0$. Adding operator reach_g2 $= \langle\{a\}, \{g_2\}, \{a\}\rangle$ makes g_2 achievable. However, no reachable state contains both g_1 and g_2, because both operators delete a, which is the only initial proposition. So g_1 and g_2 become mutex, increasing $\mathcal{I}_{\text{MX}}^{\text{scope}}(\Pi)$ from 0 to 2. □

Proposition 8. *$\mathcal{I}_{MX}^{scope}$ satisfies Goal Monotonicity.*

Proof. Adding goal g to G can only increase the set of potential mutex pairs. If g is mutex with existing goals, $\mathcal{I}_{\text{MX}}^{\text{scope}}$ increases. Otherwise it remains unchanged. □

Proposition 9. *$\mathcal{I}_{GS}^{scope}$ does not satisfy Planning Consistency.*

Proof. Counterexample: In the Light Switch problem there are no sequencing conflicts since actions are reversible, so $\mathcal{I}_{\text{GS}}^{\text{scope}}(\Pi) = 0$, yet the problem is unsolvable because the goals are mutex. □

Proposition 10. *$\mathcal{I}_{GS}^{scope}$ does not satisfy Operator Monotonicity.*

Proof. Counterexample: Consider $P = \{a, g_1, g_2\}$, $I = \{a\}$, $G = \{g_1, g_2\}$, and initial operator set $O = \{\text{reach\_g1}\}$ where reach_g1 $= \langle\{a\}, \{g_1\}, \{a\}\rangle$. Goal g_1 is achievable but g_2 is not, so no sequencing conflict exists. Adding operator reach_g2 $= \langle\{a\}, \{g_2\}, \{a\}\rangle$ makes g_2 achievable. However, from either state $\{g_1\}$ or $\{g_2\}$, neither operator is applicable since both require $\{a\}$, which has been deleted. Thus both (g_1, g_2) and (g_2, g_1) become sequencing conflicts, so g_1 and g_2 each participate, increasing $\mathcal{I}_{\text{GS}}^{\text{scope}}(\Pi)$ from 0 to 2. □

Proposition 11. *$\mathcal{I}_{GS}^{scope}$ satisfies Goal Monotonicity.*

Proof. Adding goal g to G can only increase the set of potential sequencing conflicts. New conflicts of the form (g, g') or (g', g) may emerge. Existing conflicts are unaffected. □

Each structure variant also fails Planning Consistency, by the same counterexamples used for its scope counterpart. The Light Switch problem has no unreachable propositions and no sequencing conflicts, so $\mathcal{I}_{\text{UR}}^{\text{struct}}(\Pi) = \mathcal{I}_{\text{GS}}^{\text{struct}}(\Pi) = 0$. The Locked Door problem has no mutex pairs, so $\mathcal{I}_{\text{MX}}^{\text{struct}}(\Pi) = 0$. Both problems are unsolvable. The structure variants otherwise satisfy the same postulates as their scope counterparts:

Proposition 12. *$\mathcal{I}_{UR}^{struct}$ satisfies Operator Monotonicity.*

Proof. Adding an operator can only expand S^h_{reach}, potentially making previously unreachable propositions reachable. This can only decrease or maintain the count of unreachable propositions in P. □

Proposition 13. *$\mathcal{I}_{MX}^{struct}$ does not satisfy Operator Monotonicity.*

Proof. The same counterexample as for $\mathcal{I}^{\text{scope}}_{\text{MX}}$ applies. The added operator creates a new mutex pair $\{g_1, g_2\}$, increasing $\mathcal{I}^{\text{struct}}_{\text{MX}}(\Pi)$ from 0 to 1. □

Proposition 14. *$\mathcal{I}_{GS}^{struct}$ does not satisfy Operator Monotonicity.*

Proof. The same counterexample as for $\mathcal{I}^{\text{scope}}_{\text{GS}}$ applies. Adding reach_g2 creates sequencing conflicts (g_1, g_2) and (g_2, g_1), increasing $\mathcal{I}^{\text{struct}}_{\text{GS}}(\Pi)$ from 0 to 2. □

All three structure variants satisfy Goal Monotonicity. Adding a goal g to G increases the domain over which unreachable propositions, mutex pairs, and sequencing conflicts are counted, and existing counts are unaffected since S^h_{reach} depends only on I, O, and h.

Table 3 summarizes postulate compliance. The observation that no single measure satisfies Planning Consistency motivates the multi-faceted diagnostic approach adopted in this thesis, paralleling Ulbricht et al. [40], who found that multiple measures with different postulate profiles were necessary to adequately characterize inconsistency in ASP.

	$\mathcal{I}_{\text{UR}}$		$\mathcal{I}_{\text{MX}}$		$\mathcal{I}_{\text{GS}}$	
Postulate	scope	struct	scope	struct	scope	struct
Planning Consistency	×	×	×	×	×	×
Operator Monotonicity	✓	✓	×	×	×	×
Goal Monotonicity	✓	✓	✓	✓	✓	✓

Table 3: Postulate compliance of proposed planning inconsistency measures.

4.5.2. Measure Relationships

Beyond postulate compliance, the following propositions establish how the three measures relate to each other. Rather than detecting independent failure modes, the measures exhibit a containment structure: sequencing conflicts are a strict subset of mutex relationships, while unreachability is independent of both.

Proposition 15 (Measure Relationships)**.** *For any planning problem* Π *and horizon* h*, the proposed measures exhibit the following relationships:*

1. *If goal* g *is unreachable at horizon* h*, then* g *does not participate in any mutex relationship or sequencing conflict at horizon* h
2. *If* (g_1, g_2) *is a sequencing conflict at horizon* h *(with* g_2 *achievable), then* g_1 *and* g_2 *are mutex at horizon* h
3. *Goals can be mutex at horizon* h *without having any sequencing conflict at horizon* h

Proof. 1. Both mutex (Definition 22) and sequencing conflict (Definition 25) require participating goals to be achievable within S^h_{reach}. If g is unreachable at horizon h, it fails this condition.

2. Suppose (g_1, g_2) is a sequencing conflict at horizon h with g_2 achievable. By Definition 25, g_1 is achievable and no execution trace of length at most h achieves g_1 at time i and g_2 at time $j \geq i$. In particular, no state in S^h_{reach} contains both g_1 and g_2 (which would correspond to $i = j$). Combined with both goals being achievable, this satisfies Definition 22.

3. Light Switch: Goals $\{\text{on}, \text{off}\}$ are mutex (both achievable, never co-occur) but have no sequencing conflict (from any *on*-state, *off* is reachable via *turn_off*, and vice versa).

□

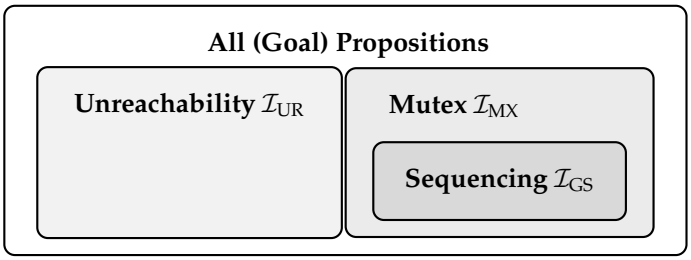

Figure 3: Diagnostic measure relationships.

The scope and structure variants of each measure are also formally related:

Proposition 16 (Scope-Structure Bounds)**.** *For any planning problem* Π *and horizon* h*:*

1. $\mathcal{I}_{UR}^{scope}(\Pi, h) \leq \mathcal{I}_{UR}^{struct}(\Pi, h) \leq |P|$
2. $\mathcal{I}_{MX}^{struct}(\Pi, h) \leq \binom{\mathcal{I}_{MX}^{scope}(\Pi, h)}{2}$
3. $\mathcal{I}_{GS}^{struct}(\Pi, h) \leq \mathcal{I}_{GS}^{scope}(\Pi, h) \times (\mathcal{I}_{GS}^{scope}(\Pi, h) - 1)$

Proof. 1. Unreachable goals are a subset of all unreachable propositions, which are a subset of P.

2. The maximum number of unordered pairs among n goals is $\binom{n}{2}$, where $n = \mathcal{I}_{\text{MX}}^{\text{scope}}(\Pi, h)$.

3. The maximum number of ordered pairs among $\mathcal{I}_{\text{GS}}^{\text{scope}}(\Pi, h)$ goals is $n(n-1)$ where $n = \mathcal{I}_{\text{GS}}^{\text{scope}}(\Pi, h)$.

□

4.5.3. Computational Complexity

This section analyzes the computational complexity of the proposed measures in terms of the complexity classes introduced in Section 2.3. Determining plan existence, PLANSAT, for propositional STRIPS is PSPACE-complete [6]. All three measures depend on reachability analysis over the planning state space, and this structural connection to PLANSAT determines their decision-problem complexity.

Following Thimm and Wallner [39] and Corea et al. [7], four computational problems are defined for each measure $\mathcal{I} \in \{\mathcal{I}_{\text{UR}}, \mathcal{I}_{\text{MX}}, \mathcal{I}_{\text{GS}}\}$.

Definition 31 (Decision and Function Problems)**.** Let $\Pi = \langle P, O, I, G \rangle$ be a planning problem and let $k \in \mathbb{N}_0$. Write $\mathcal{I}(\Pi)$ for the converged measure

$$\mathcal{I}(\Pi) = \lim_{h \to \infty} \mathcal{I}(\Pi, h).$$

The following problems are defined for each measure $\mathcal{I}$:

1. UPPER$_{\mathcal{I}}$: is $\mathcal{I}^{\text{scope}}(\Pi) \leq k$?
2. LOWER$_{\mathcal{I}}$: is $\mathcal{I}^{\text{scope}}(\Pi) \geq k$?
3. EXACT$_{\mathcal{I}}$: is $\mathcal{I}^{\text{scope}}(\Pi) = k$?
4. VALUE$_{\mathcal{I}}$: compute $\mathcal{I}^{\text{scope}}(\Pi)$.

For the horizon-bounded variants $\mathcal{I}(\Pi, h)$ with h given as part of the input, the PSPACE membership arguments below apply unchanged: any reachable state admits an acyclic witness path visiting at most $2^{|P|}$ distinct states, so the effective search depth is bounded by $\min(h, 2^{|P|})$ and fits in a polynomial-size counter. PSPACE-hardness extends to the horizon-bounded variants by setting $h = 2^{|P|}$ in the reductions below, which is encodable in $|P|$ bits.

Analogous problems are defined for $\mathcal{I}^{\text{struct}}$. The scope variants iterate over G or $G \times G$, while the structure variants iterate over P, unordered pairs from G, or ordered pairs from G, respectively. Since $|G| \leq |P|$ and every iteration domain is polynomial in the input size, the membership arguments below apply verbatim to the structure variants. The proofs are therefore stated for the scope variants, and

the matching hardness of the structure variants is established collectively in Proposition 17.

Each completeness result below establishes membership and hardness as defined in Section 2.3.

Theorem 2. *UPPER$_{\mathcal{I}_{UR}}$ is PSPACE-complete.*

Proof. Membership. Checking whether $\mathcal{I}_{\text{UR}}^{\text{scope}} \leq k$ is equivalent to verifying that at least $|G| - k$ goal propositions are reachable. For each $g \in G$, the question "does there exist a reachable state s with $g \in s$?" can be decided in NPSPACE as follows. A nondeterministic machine guesses a sequence of operator applications starting from I, storing only the current state and a step counter in polynomial space. Since the state space contains at most $2^{|P|}$ distinct states, any reachable state can be reached via a path that never revisits a state. If a path visits the same state twice, the cycle between the two visits can be removed, yielding a shorter path that still reaches the target. Therefore, the machine can accept if a state containing g is reached within $2^{|P|}$ steps, and reject otherwise. By Savitch's theorem (Section 2.3), NPSPACE = PSPACE. Since $|G| \leq |P|$, iterating this check over all goals adds only polynomial overhead, and PSPACE is closed under polynomial-time composition. Therefore UPPER$_{\mathcal{I}_{\text{UR}}} \in$ PSPACE.

Hardness. We reduce from PLANSAT. Given a PLANSAT instance $\Pi = \langle P, O, I, G \rangle$, construct $\Pi' = \langle P', O', I, G' \rangle$ where:

- $P' = P \cup \{g^*\}$ for a fresh proposition g^*,
- $G' = \{g^*\}$,
- $O' = O \cup \{o^*\}$ where $o^* = \langle G, \{g^*\}, \emptyset \rangle$.

The operator o^* is applicable exactly when all original goals are satisfied, producing g^*. Thus g^* is reachable in Π' if and only if Π has a plan. Therefore $\mathcal{I}_{\text{UR}}^{\text{scope}}(\Pi') = 0$ if and only if Π is solvable. Deciding UPPER$_{\mathcal{I}_{\text{UR}}}$ with $k = 0$ solves PLANSAT, establishing PSPACE-hardness. □

Corollary 1. *For $\mathcal{I}_{UR}$, LOWER and EXACT are PSPACE-complete, and VALUE is in FPSPACE.*

Proof. LOWER$_{\mathcal{I}_{\text{UR}}}$ with threshold k is the complement of UPPER$_{\mathcal{I}_{\text{UR}}}$ with threshold $k - 1$, placing it in PSPACE since PSPACE = co-PSPACE (Section 2.3). For hardness, the reduction in Theorem 2 yields $\mathcal{I}_{\text{UR}}^{\text{scope}}(\Pi') \in \{0, 1\}$ with value 1 iff Π is unsolvable, so LOWER$_{\mathcal{I}_{\text{UR}}}$ with $k = 1$ decides the complement of PLANSAT, which is PSPACE-complete. Therefore LOWER$_{\mathcal{I}_{\text{UR}}}$ is PSPACE-complete.

EXACT$_{\mathcal{I}_{\text{UR}}}$ can be decided by checking both $\mathcal{I}_{\text{UR}}^{\text{scope}} \leq k$ and $\mathcal{I}_{\text{UR}}^{\text{scope}} \geq k$. Hardness follows from the same reduction as Theorem 2, since $\mathcal{I}_{\text{UR}}^{\text{scope}}(\Pi') \in \{0, 1\}$, and deciding EXACT$_{\mathcal{I}_{\text{UR}}}$ with $k = 0$ solves PLANSAT. Therefore PSPACE-complete.

VALUE$_{\mathcal{I}_{\text{UR}}}$ can be computed via binary search with $O(\log |G|)$ calls to UPPER$_{\mathcal{I}_{\text{UR}}}$. Polynomially many calls to a PSPACE subroutine remain in FPSPACE. □

In the propositional setting of Thimm and Wallner [39], LOWER is coNP-complete rather than NP-complete, and EXACT lands in D^p, the class of problems decidable by the conjunction of an NP and a coNP problem, believed to be strictly larger than NP$\cap$coNP. At the PSPACE level, PSPACE $=$ co-PSPACE collapses both distinctions. PSPACE is a deterministic class, so a machine deciding membership also decides the complement by negating its output, and the conjunction of two PSPACE problems remains in PSPACE. The broader consequences for the complexity landscape are discussed after Corollary 3.

Theorem 3. *UPPER$_{\mathcal{I}_{MX}}$ is PSPACE-complete.*

Proof. Membership. Checking whether $\mathcal{I}_{\text{MX}}^{\text{scope}} \leq k$ requires verifying that at most k goals participate in mutex relationships. For each pair $(g_1, g_2) \in G \times G$ with $g_1 \neq g_2$, the mutex check (Definition 22) involves verifying that each goal is individually reachable and that no reachable state contains both. Individual reachability is in PSPACE by the same nondeterministic argument as for $\mathcal{I}_{\text{UR}}$. The non-coexistence check is in co-PSPACE $=$ PSPACE. Since PSPACE is closed under Boolean combination and there are at most $|G|^2$ pairs, the overall problem is in PSPACE.

Hardness. We reduce from PLANSAT. Given $\Pi = \langle P, O, I, G\rangle$, construct $\Pi' = \langle P', O', I', G'\rangle$ where $P' = P \cup \{g_1, g_2, \mathit{flag}\}$ for three fresh propositions, $I' = I \cup \{g_1\}$, $G' = \{g_1, g_2\}$, and $O' = O \cup \{o_f, o_a, o_m\}$ with:

$$o_f = \langle \emptyset,\ \{\mathit{flag}\},\ \{g_1\}\rangle$$
$$o_a = \langle \{\mathit{flag}\},\ \{g_2\},\ \emptyset\rangle$$
$$o_m = \langle G,\ \{g_2\},\ \emptyset\rangle$$

The original operators in O affect only propositions in P and never modify g_1, g_2, or *flag*. Goal g_1 is achievable since it is in I'. Goal g_2 is always achievable via o_f followed by o_a.

If Π is solvable, a plan for Π reaches a state satisfying G, from which o_m is applicable, producing g_2 without deleting g_1. The resulting state contains both g_1 and g_2, so the goals are not mutex, and $\mathcal{I}_{\text{MX}}^{\text{scope}}(\Pi') = 0$.

If Π is unsolvable, o_m is never applicable because its precondition G is never satisfied. The only path to g_2 is $o_f \rightarrow o_a$, but o_f deletes g_1 and no operator in O' re-adds it. No reachable state contains both g_1 and g_2, so the goals are mutex, and $\mathcal{I}_{\text{MX}}^{\text{scope}}(\Pi') = 2$.

Therefore UPPER$_{\mathcal{I}_{\text{MX}}}$ with $k = 0$ decides PLANSAT, establishing PSPACE-hardness. □

Corollary 2. *For $\mathcal{I}_{MX}$, LOWER and EXACT are PSPACE-complete, and VALUE is in FPSPACE.*

Proof. By the same arguments as Corollary 1, applied to the reduction from Theorem 3. □

Theorem 4. *UPPER$_{\mathcal{I}_{GS}}$ is PSPACE-complete.*

Proof. Membership. The sequencing conflict check (Definition 25) requires that both goals are individually reachable and that no execution trace from I achieves g_1 and subsequently g_2. At the converged horizon, the no-following condition is equivalent to verifying that for every reachable state s with $g_1 \in s$, no state in REACH(s) contains g_2. This local reachability query is itself in PSPACE by the same nondeterministic search argument. Checking this condition for all g_1-states is a universal property, placing it in co-PSPACE = PSPACE. Since all sub-checks are in PSPACE and their combination involves polynomial overhead, the overall problem remains in PSPACE.

Hardness. The reduction from Theorem 3 also establishes hardness for $\mathcal{I}_{\text{GS}}$. In the constructed instance Π', when Π is unsolvable, no operator in O' adds g_1 because the only source of g_1 is the initial state, and o_f deletes it irreversibly. Thus from any g_2-state, which can only be reached via $o_f \to o_a$, g_1 is unreachable, making (g_2, g_1) a sequencing conflict. Conversely, when Π is solvable, o_m produces a state where both goals co-occur, so no sequencing conflict exists. Therefore UPPER$_{\mathcal{I}_{\text{GS}}}$ with $k = 0$ decides PLANSAT. □

Corollary 3. *For $\mathcal{I}_{GS}$, LOWER and EXACT are PSPACE-complete, and VALUE is in FPSPACE.*

Proof. By the same arguments as Corollary 1, applied to the reduction from Theorem 4. □

Proposition 17 (Complexity of the Structure Variants)**.** *For each of $\mathcal{I}_{UR}$, $\mathcal{I}_{MX}$, and $\mathcal{I}_{GS}$, the structure variants of UPPER, LOWER, and EXACT are PSPACE-complete, and VALUE is in FPSPACE.*

Proof. Membership. The structure variants invoke the same reachability subroutines as the scope variants over iteration domains polynomial in the input size, so the membership arguments of Theorems 2, 3, and 4 apply unchanged.

Hardness. For $\mathcal{I}_{\text{MX}}$ and $\mathcal{I}_{\text{GS}}$, the instance Π' of Theorems 3 and 4 contains, in the unsolvable case, exactly one structural artifact, namely the mutex goal pair $\{g_1, g_2\}$ or the ordered sequencing-conflict pair (g_2, g_1), and none in the solvable case. Hence $\mathcal{I}^{\text{struct}}(\Pi') \in \{0, 1\}$, and UPPER with $k = 0$ decides PLANSAT.

For $\mathcal{I}_{\text{UR}}$, whose structure variant counts all unreachable propositions rather than only unreachable goal propositions, we adapt the reduction of Theorem 2. Given a PLANSAT instance $\Pi = \langle P, O, I, G\rangle$, construct

$$\Pi'' = \langle P \cup \{g^*\}, O \cup \{o^*\} \cup \{o_p : p \in P\}, I, \{g^*\}\rangle,$$

with $o^* = \langle G, \{g^*\}, \emptyset\rangle$ and $o_p = \langle \{g^*\}, \{p\}, \emptyset\rangle$ for each $p \in P$. Each o_p requires g^*, and g^* requires all original goals. The added operators therefore cannot influence whether G is first achieved, so the solvability of the embedded instance is preserved. If Π is solvable, g^* is reachable and every o_p then makes its proposition reachable,

so $\mathcal{I}_{\text{UR}}^{\text{struct}}(\Pi'') = 0$. If Π is unsolvable, g^* is never reachable and $\mathcal{I}_{\text{UR}}^{\text{struct}}(\Pi'') \geq 1$. The construction is computable in polynomial time, so deciding the structure variant of $\text{UPPER}_{\mathcal{I}_{\text{UR}}}$ with $k = 0$ solves PLANSAT, establishing PSPACE-hardness.

The LOWER, EXACT, and VALUE problems follow by the same arguments as in Corollaries 1, 2, and 3. □

Table 4 summarizes these results.

Problem	$\mathcal{I}_{\text{UR}}$	$\mathcal{I}_{\text{MX}}$	$\mathcal{I}_{\text{GS}}$
UPPER	PSPACE-c.	PSPACE-c.	PSPACE-c.
LOWER	PSPACE-c.	PSPACE-c.	PSPACE-c.
EXACT	PSPACE-c.	PSPACE-c.	PSPACE-c.
VALUE	in FPSPACE	in FPSPACE	in FPSPACE

Table 4: Complexity of decision and function problems for the three profiles.

The uniform PSPACE tier observed above contrasts with the multi-tier structure for inconsistency measures in other formalisms. Thimm and Wallner [39] show that propositional inconsistency measures span three distinct tiers, ranging from NP-complete through Σ_2^p- or Π_2^p-complete to $\# \cdot$ coNP-complete for counting variants. This stratification arises because the base satisfiability problem is NP-complete, and different measure definitions add varying computational overhead above that base. Corea et al. [7] analyze measures for declarative process specifications over LTL_{ff}, where satisfiability is also NP-complete. Their time-sensitive inconsistency measures consequently land at the first level of the polynomial hierarchy, yielding NP-complete, coNP-complete, D^p, and $\text{FP}^{\text{NP}[\log n]}$ problems.

For classical planning, PLANSAT is PSPACE-complete, and all three proposed measures reduce to polynomial compositions of reachability queries. The polynomial overhead from different measure definitions cannot push the complexity above PSPACE, because PSPACE is closed under polynomial time reductions. The result is a flat complexity landscape where all decision problems for all three measures are PSPACE-complete. This uniformity reflects a general principle. The complexity of inconsistency measures inherits from the base reasoning problem of the target formalism, and when that base problem is already PSPACE-complete, the polynomial differences between measure definitions are subsumed.

The preceding results characterize the inherent complexity of the decision problems. The following propositions complement them with explicit time and space bounds for the enumeration-based implementation.

Proposition 18 (Reachability Complexity)**.** *Computing the reachable state set S_{reach} for a planning problem $\langle P, O, I, G\rangle$ requires $O(2^{|P|} \cdot |O| \cdot |P|)$ time and $O(2^{|P|} \cdot |P|)$ space.*

Proof. The reachable state set has at most $2^{|P|}$ elements. The fixpoint computation maintains the set of discovered states and processes each state at most once. For

each discovered state, up to $|O|$ operators are tested for applicability, and each such test requires $O(|P|)$ for precondition checking and effect computation. Since at most $2^{|P|}$ states are ever discovered, the total time is $O(2^{|P|} \cdot |O| \cdot |P|)$. Each state requires $O(|P|)$ space for its representation, giving $O(2^{|P|} \cdot |P|)$ total space. □

Proposition 19 (Unreachability Profile Complexity)**.** *Given S_{reach}, computing $\mathcal{I}_{UR}^{scope}$ and $\mathcal{I}_{UR}^{struct}$ requires $O(|S_{reach}| \cdot |P|)$ time and $O(|P|)$ additional space.*

Proof. For each proposition $p \in P$, check whether p appears in any state $s \in S_{\text{reach}}$. This requires one pass through all states. □

Proposition 20 (Goal Mutex Profile Complexity)**.** *Given S_{reach}, computing $\mathcal{I}_{MX}^{scope}$ and $\mathcal{I}_{MX}^{struct}$ requires $O(|S_{reach}| \cdot |G|^2)$ time and $O(|G|^2)$ additional space.*

Proof. For each goal pair (g_1, g_2), verify achievability of each goal ($O(|S_{\text{reach}}| \cdot |G|)$) and check no state contains both ($O(|S_{\text{reach}}| \cdot |G|^2)$). Results stored in a $|G| \times |G|$ matrix. □

Proposition 21 (Goal Sequencing Conflict Profile Complexity)**.** *Given S_{reach}, computing $\mathcal{I}_{GS}^{scope}$ and $\mathcal{I}_{GS}^{struct}$ requires $O(|G|^2 \cdot |S_{reach}|^2 \cdot |O|)$ time and $O(|S_{reach}|)$ additional space.*

Proof. Since $\text{REACH}(s) \subseteq S_{\text{reach}}$ for any $s \in S_{\text{reach}}$, local reachability is computed as graph traversal within the pre-computed state space rather than full reachability. For each goal pair (g_1, g_2), check all g_1-states. Each g_1-state requires traversing at most $|S_{\text{reach}}|$ states with $|O|$ transitions each. There are $O(|G|^2)$ pairs and up to $|S_{\text{reach}}|$ starting states per pair. □

Table 5 summarizes these results.

Measure	Time Complexity	Space Complexity
S_{reach}	$O(2^{\|P\|} \cdot \|O\| \cdot \|P\|)$	$O(2^{\|P\|} \cdot \|P\|)$
$\mathcal{I}_{\text{UR}}$ (given S_{reach})	$O(\|S_{\text{reach}}\| \cdot \|P\|)$	$O(\|P\|)$
$\mathcal{I}_{\text{MX}}$ (given S_{reach})	$O(\|S_{\text{reach}}\| \cdot \|G\|^2)$	$O(\|G\|^2)$
$\mathcal{I}_{\text{GS}}$ (given S_{reach})	$O(\|G\|^2 \cdot \|S_{\text{reach}}\|^2 \cdot \|O\|)$	$O(\|S_{\text{reach}}\|)$

Table 5: Computational complexity of the profiles.

The exponential dependence on $|P|$ reflects the inherent complexity of planning. However, real planning problems often have much smaller reachable state spaces than the theoretical maximum. For example, the Locked Door scenario has $|P| = 4$ but $|S_{\text{reach}}| = 1$. Additionally, all three measures share the S_{reach} computation, amortizing this cost when computing multiple profiles. The Goal Sequencing Conflict Profile has higher complexity than the other measures due to requiring multiple local reachability computations. For expensive problems, the Unreachability and Mutex Profiles can be computed independently as partial diagnostics.

The complexity analysis above assumes explicit state space enumeration. ASP-based implementations may exhibit different practical performance characteristics due to several factors. First, modern ASP solvers employ conflict-driven learning, unit propagation, and backjumping, reducing inferences to constraint propagation and enabling efficient search space traversal [19]. Second, the declarative nature of ASP encodings allows solvers to exploit problem structure automatically.

Consequently, the theoretical worst-case bounds may significantly overestimate practical computation times. The empirical evaluation (Section 6) will assess actual scalability on benchmark problems, providing complementary evidence to the theoretical complexity analysis. Kuhlmann et al. [29] demonstrate that ASP-based implementations significantly outperform both SAT-based and naive baseline approaches for computing propositional inconsistency measures, suggesting that similar advantages may apply to the planning-based measures proposed here.

4.6. Discussion

This section interprets the formal results from Section 4.5, showing how the measure relationships and scope-structure bounds translate into actionable diagnostic information. All three measures share identical decision-problem complexity (Theorems 2–4), so the choice of which profiles to compute should be driven by diagnostic needs rather than computational cost. The practical time-complexity differences documented in Table 5 remain relevant for explicit enumeration approaches, where the Goal Sequencing Conflict Profile requires additional local reachability computations. The section then discusses normalization considerations for practical deployment.

4.6.1. Diagnostic Interpretation

The measure relationships established in Proposition 15 reflect increasing diagnostic specificity. Unreachability identifies goals that cannot be achieved at all. Mutex identifies achievable goals that cannot coexist. Sequencing identifies mutex goals where the exclusion is irreversible. Figure 3 illustrates this containment structure. Two cases are particularly informative:

- **Mutex without sequencing** (e.g., Light Switch): Goals exclude each other but actions are reversible, so the conflict is symmetric and neither goal is more responsible for unsolvability.
- **Mutex with sequencing** (e.g., Trust and Travel): Goals exclude each other *and* achieving specific goals creates permanent dead states. The sequencing measure identifies which goal achievement is responsible, providing actionable information for adding reversal operators.

State space analysis (unreachability, mutex) determines what states exist, while trajectory analysis (sequencing) determines what orderings of goal achievements

are possible. For irreversible problems, all three measures report non-zero values, with sequencing providing directional information about which goal achievements lead to dead states.

The scope-structure bounds (Proposition 16) enable further diagnostic interpretation through ratio analysis:

- **Unreachability depth**: When $\mathcal{I}_{\text{UR}}^{\text{scope}} > 0$, the ratio $(\mathcal{I}_{\text{UR}}^{\text{struct}} - \mathcal{I}_{\text{UR}}^{\text{scope}})/\mathcal{I}_{\text{UR}}^{\text{scope}}$ indicates cascading dependency depth. A ratio of 0 means goals are directly unreachable, with no intermediate propositions blocked. Larger ratios indicate deeper precondition chains.

- **Mutex density**: If $\mathcal{I}_{\text{MX}}^{\text{struct}} = \binom{\mathcal{I}_{\text{MX}}^{\text{scope}}}{2}$, the mutex goals form a complete clique where every pair is mutually exclusive, as in Traffic Light. Ratios below 1 indicate sparser conflict structures with isolated mutex pairs.

- **Sequencing symmetry**: If $\mathcal{I}_{\text{GS}}^{\text{struct}} = \mathcal{I}_{\text{GS}}^{\text{scope}} \times (\mathcal{I}_{\text{GS}}^{\text{scope}} - 1)$, all sequencing conflicts are bidirectional: achieving any involved goal blocks all others, and vice versa. Ratios near 0.5 suggest predominantly unidirectional conflicts. For goal sets with three or more conflict participants, lower ratios indicate single-source dominant patterns where one goal blocks many others but not conversely.

Table 6 demonstrates these relationships across the test scenarios. For problems exhibiting only unreachability, only $\mathcal{I}_{\text{UR}}$ yields non-zero values. For reversible mutex problems, only $\mathcal{I}_{\text{MX}}$ yields non-zero values. Irreversible problems yield non-zero values for both $\mathcal{I}_{\text{MX}}$ and $\mathcal{I}_{\text{GS}}$, consistent with Proposition 15.

Scenario	$\mathcal{I}_{\text{UR}}$		$\mathcal{I}_{\text{MX}}$		$\mathcal{I}_{\text{GS}}$	
	scope	struct	scope	struct	scope	struct
Locked Door	1	3	0	0	0	0
Bank Vault	1	7	0	0	0	0
Light Switch	0	0	2	1	0	0
Traffic Light	0	0	3	3	0	0
Trust and Travel	0	0	2	1	2	1
Rival Alliances	0	0	2	1	2	2

Table 6: Measure values across test scenarios.

The complete diagnostic profile for a planning problem is the 6-tuple:

$$(\mathcal{I}_{\text{UR}}^{\text{scope}}(\Pi, h), \mathcal{I}_{\text{UR}}^{\text{struct}}(\Pi, h), \mathcal{I}_{\text{MX}}^{\text{scope}}(\Pi, h), \mathcal{I}_{\text{MX}}^{\text{struct}}(\Pi, h), \mathcal{I}_{\text{GS}}^{\text{scope}}(\Pi, h), \mathcal{I}_{\text{GS}}^{\text{struct}}(\Pi, h))$$

Table 6 lists the converged profiles $\text{Profile}(\Pi)$ for all test scenarios.

Each scenario in Table 6 exhibits a characteristic profile signature. Locked Door illustrates dependency-structure failure with depth 2, where one unreachable goal

proposition appears among three unreachable propositions in total. Bank Vault shares the same scope value at depth 6, reflecting three converging precondition chains. The two reversible-mutex scenarios differ in clique completeness: Light Switch yields a single mutex pair, while Traffic Light closes a complete 3-clique with all $\binom{3}{2} = 3$ pairs mutex. Trust and Travel and Rival Alliances share mutex structure ($\mathcal{I}_{\text{MX}}^{\text{struct}} = 1$) but separate by sequencing direction: Trust and Travel exhibits unidirectional sequencing ($\mathcal{I}_{\text{GS}}^{\text{struct}} = 1$) and Rival Alliances exhibits bidirectional sequencing ($\mathcal{I}_{\text{GS}}^{\text{struct}} = 2$). This contrast illustrates how trajectory analysis (sequencing) provides finer-grained diagnostic information beyond state space analysis (unreachability, mutex). These interpretive observations rely on the absolute counts produced by the measures, raising the question of whether such counts should be normalized for cross-problem comparison.

4.6.2. Normalization Considerations

The three proposals presented above report absolute inconsistency values: counts of unreachable propositions, mutex pairs, and sequencing conflict pairs. A practical question is whether these measures should be normalized relative to problem characteristics such as the total number of goal propositions or the size of the proposition space.

Besnard and Grant [2] develop relative inconsistency measures for propositional knowledge bases, demonstrating both advantages and limitations of normalization approaches. Relative measures can facilitate comparison across problems of different sizes, potentially providing a notion of percentage inconsistency analogous to test coverage metrics in software engineering.

However, several considerations suggest that absolute measures may be more appropriate for planning diagnostics. First, the severity of unsolvability is not necessarily proportional to problem size. A planning problem with 100 goal propositions where only 2 conflict may be easier to repair than a problem with 3 goal propositions where all 3 form a complete conflict graph. Second, normalization requires choosing an appropriate denominator from candidates such as the total goal-proposition count, the proposition-space size, or the maximum possible conflict count, and each choice encodes a different assumption about problem structure. Third, the proposed measures already provide multiple values that capture relative information: the ratio between unreachable goal propositions and total unreachable propositions indicates dependency depth, while the ratio between mutex goals and mutex pairs (or conflicting goals and sequencing pairs) reveals conflict structure.

For the scope of this thesis, the focus remains on absolute measures that report objective structural properties. Future work could calibrate normalization schemes against the IPC 2016 benchmark distribution to compare inconsistency trends across problem generator parameters or to establish thresholds for automated problem classification.

Having defined the measures and analyzed their postulates, relationships, and complexity, Section 5 turns to their computation, presenting the ASP-based pipeline that realizes the parameterized profiles on standard PDDL input.

5. Implementation

This section presents the ASP-based implementation of the adapted inconsistency measures from Section 4.4. The system accepts PDDL planning problems as input and, given a horizon parameter h, computes the full diagnostic profile

$$(\mathcal{I}_{\mathrm{UR}}^{\mathrm{scope}}(\Pi, h), \mathcal{I}_{\mathrm{UR}}^{\mathrm{struct}}(\Pi, h), \mathcal{I}_{\mathrm{MX}}^{\mathrm{scope}}(\Pi, h), \mathcal{I}_{\mathrm{MX}}^{\mathrm{struct}}(\Pi, h), \mathcal{I}_{\mathrm{GS}}^{\mathrm{scope}}(\Pi, h), \mathcal{I}_{\mathrm{GS}}^{\mathrm{struct}}(\Pi, h))$$

through a three-phase pipeline of PDDL translation, ASP state-space exploration via brave reasoning, and Python-based measure extraction. The horizon parameter h directly corresponds to the formal parameter in the measure definitions (Section 4.4.2).

5.1. System Architecture

The implementation follows a pipeline architecture with three distinct phases, each handled by a dedicated component.

Phase 1: PDDL Translation. The PDDL domain and problem files are translated into ASP facts using *plasp* 3 by Dimopoulos et al. [9], a standard tool from the Potassco ASP ecosystem. The translation produces a lifted ASP program where propositions are represented as boolean-valued variables and operators as parameterized actions. For benchmark domains with the `:functions` requirement, a preprocessing step strips cost-related elements because they do not affect reachability.

Phase 2: ASP Brave Reasoning. The translated ASP program is loaded alongside a custom reachability encoding and solved using Clingo 5 by Gebser et al. [19] with the `--enum-mode=brave` configuration. Under brave reasoning (Definition 14), Clingo computes the union of atoms that are true in *any* answer set, without enumerating individual answer sets. This mode is central to the implementation. Different answer sets correspond to different operator execution traces, and the union across all answer sets captures the full scope of reachable states and achievable propositions. Dedicated witness atoms record whether goal pairs can coexist in a single state or be achieved in sequence; the absence of such a witness under brave reasoning constitutes proof that no execution trace satisfies the corresponding condition, which is the detection mechanism for mutex and sequencing conflicts. The grounding and solving phases are timed separately, enabling per-phase performance analysis (Section 6.5.2).

Phase 3: Measure Extraction. A Python post-processing module collects the brave reasoning atoms and computes all six measure values through set operations. Unreachability is determined by comparing goal propositions against the set of truly reachable propositions. Mutex and sequencing conflicts are identified by the absence of the corresponding witness atoms. If brave reasoning, which considers all possible execution traces, never derives a witness, the conflict is guaranteed to hold. The details of the witness definitions and their correspondence to the formal measure definitions are given in Section 5.3.3.

The implementation is written in Python 3.12 with Clingo 5.8.0 as the sole computational dependency. Docker containerization enables consistent execution regardless of the environment, and an automated test suite ensures correctness and facilitates regression testing during development. The codebase is structured to allow both library usage and batch processing of benchmark domains, with configurable time limits to handle grounding complexity in larger problems. The complete source code, including all ASP encodings, the Python measure extraction module, and the benchmark runner, is publicly available [5].

5.2. Input Translation

The input to the system is a pair of standard PDDL files, a domain specification defining predicates and action schemas and a problem specification defining objects, the initial state, and goal conditions.

5.2.1. Adoption of plasp

An initial prototype grounded PDDL action schemas via typed Cartesian product enumeration using the `pddl` Python library by Favorito et al. [15]. Testing on the IPC 2016 benchmarks by Muise and Lipovetzky [32] revealed two limitations of this approach. The library does not support the `:constants` declaration used by several IPC domains, and naive Cartesian product grounding produces prohibitively large programs for domains that use predicates to constrain parameter interactions. For example, the *pegsol* domain produced over 395,000 ground ASP facts because the translator enumerated all position triples regardless of the constraining `in-line` predicate.

The implementation was therefore restructured to use *plasp* 3 by Dimopoulos et al. [9] for PDDL translation. The *plasp* tool, developed as part of the Potassco project, translates PDDL specifications into a lifted ASP representation where Clingo handles the grounding process. This offers two advantages. First, *plasp* correctly handles PDDL features including `:constants` and complex type hierarchies. Second, Clingo's grounding algorithm exploits the structure of ASP rules to avoid generating impossible parameter combinations.

The *plasp* output format represents propositions as boolean-valued variables and operators as parameterized actions with compound terms. Table 7 shows the cor-

respondence between the internal representation used by the reachability encoding and the *plasp* output format.

Concept	plasp format	Internal format
Initial state	`initialState(V, value(V,true))`	`init(P)`
Goal	`goal(V, value(V,true))`	`goal(P)`
Precondition	`precondition(A, V, value(V,true))`	`precond(O, P)`
Add effect	`postcondition(A, ..., value(V,true))`	`add(O, P)`
Delete effect	`postcondition(A, ..., value(V,false))`	`delete(O, P)`

Table 7: Mapping between internal and *plasp* predicates.

A bridge encoding translates between these representations. Since ASP distinguishes predicates by both name and arity, *plasp*'s `goal/2` and the internal `goal/1` coexist without conflict.

```
1 init(V) :- initialState(V, value(V, true)).
2 goal(V) :- goal(V, value(V, true)).
3 precond(A, V) :- precondition(A, V, value(V, true)).
4 neg_precond(A, V) :- precondition(A, V, value(V, false)).
5 add(A, V) :- postcondition(A, effect(unconditional),
6                             V, value(V, true)).
7 delete(A, V) :- postcondition(A, effect(unconditional),
8                                V, value(V, false)).
```

Listing 1: Bridge encoding mapping plasp to internal format.

The following example illustrates the complete translation for a small planning problem that exercises every bridge rule.

Example 8 (Bridge Translation)**.** Consider a planning problem with two operators. The `unlock` operator requires being at the door and the door being locked, then removes the lock. The `enter` operator requires being at the door and the door *not* being locked, then adds `in_room` and removes `at_door`. Applying the bridge encoding from Listing 1 to the *plasp* output, following the mapping in Table 7, yields the following internal representation:

```
1  % From initialState/2 with value(..., true)
2  init(at_door).   init(locked).
3  % From goal/2 with value(..., true)
4  goal(in_room).
5  % From precondition/3 with value(..., true)
6  precond(unlock, at_door).   precond(unlock, locked).
7  precond(enter, at_door).
8  % From precondition/3 with value(..., false)
9  neg_precond(enter, locked).
10 % From postcondition/4 with value(..., true)
11 add(enter, in_room).
```

```
12 % From postcondition/4 with value(..., false)
13 delete(unlock, locked).   delete(enter, at_door).
```

Listing 2: Internal facts derived by the bridge encoding.

Note how *plasp*'s `value(..., false)` terms produce two semantically distinct internal predicates: `neg_precond(enter, locked)` encodes that `enter` requires `locked` to be false before application, while `delete(unlock, locked)` encodes that `unlock` makes `locked` false as an effect. This distinction is central to the reachability encoding: negative preconditions restrict operator applicability (Section 5.3), while delete effects drive state change and are the root cause of mutex and sequencing conflicts.

5.2.2. Input Preprocessing

Several IPC 2016 domains declare action costs through the `:functions` requirement, which *plasp* does not support. Action costs affect only plan quality, not plan existence. Since the inconsistency measures defined in this thesis operate on reachability and achievability, removing cost annotations is semantically safe. A preprocessing step therefore removes the following cost-related elements:

- the `:action-costs` requirement,
- `(:functions ...)` declarations and `(:metric ...)` specifications,
- `(increase ...)` and `(decrease ...)` effect terms,
- numeric function value assignments.

After preprocessing, domains such as *bag-transport* and *over-nomystery* translate successfully.

The two IPC domains *bag-barman* and *tetris* use the `:equality` requirement with variable comparison expressions `(= ?x ?y)` in preconditions, which is not supported by *plasp*. These domains remain incompatible and are excluded from the evaluation.

5.3. Encoding for Measure Computation

The core of the implementation is a pair of ASP encodings that derive proposition and operator sets from the translated facts and perform horizon-bounded state-space exploration with brave reasoning. Following standard practice in the ASP literature [31, 19], the encodings are presented directly, as the declarative nature of ASP rules makes them serve both as formal specification and executable program.

5.3.1. Planning Representation

The planning encoding derives the proposition and operator sets from the input facts:

```
1  prop(P) :- init(P).
2  prop(P) :- add(_, P).
3  prop(P) :- delete(_, P).
4  prop(P) :- precond(_, P).
5  prop(P) :- goal(P).
6
7  operator(O) :- precond(O, _).
8  operator(O) :- add(O, _).
9
10 #show goal/1.
11 #show prop/1.
12 #show operator/1.
```

Listing 3: Proposition and operator derivation.

5.3.2. State-Space Exploration

The main computation performs horizon-bounded state-space exploration. The encoding models time-indexed state evolution where `holds(P, T)` indicates that proposition P is true at time step T:

```
1  time(0..horizon).
2  holds(P, 0) :- init(P).
3
4  % Operator applicability with positive
5  % and negative preconditions
6  neg_precond_violated(O, T) :-
7      neg_precond(O, P), holds(P, T), time(T).
8  applicable(O, T) :- operator(O), time(T), T < horizon,
9      not neg_precond_violated(O, T),
10     holds(P, T) : precond(O, P).
11
12 % Non-deterministic operator selection
13 { apply(O, T) : applicable(O, T) } 1 :- time(T), T < horizon.
14
15 % State transition with frame axiom
16 deleted(P, T) :- apply(O, T), delete(O, P).
17 holds(P, T+1) :- apply(O, T), add(O, P).
18 holds(P, T+1) :- holds(P, T), time(T), T < horizon,
19     not deleted(P, T).
```

Listing 4: State-space exploration encoding.

The choice rule on line 13 selects at most one applicable operator at each time step. Different selections produce different answer sets, each corresponding to a distinct execution trace through the state space. The frame axiom (line 19) ensures that propositions persist unless explicitly deleted.

5.3.3. Brave Reasoning for Witness Collection

The key insight of the implementation is the use of Clingo's brave reasoning mode (`-enum-mode=brave`). Under this configuration, Clingo computes the union of atoms that are true in at least one answer set, without enumerating individual answer sets. This is essential for efficiency. The number of possible execution traces grows exponentially with the horizon, but the brave consequences can be computed without explicit enumeration.

The choice of brave over cautious reasoning is deliberate. Brave reasoning computes the existential union BRAVE(Π) (Definition 14) over the horizon-bounded encoding, so an atom appears in the result if and only if at least one answer set contains it. For conflict detection, the converse is what matters. If a witness atom is *absent* from the brave consequences, then no answer set contains it, which provides a universal proof that the corresponding condition is unsatisfiable across all execution traces. Cautious reasoning would yield atoms true in *every* answer set, which is the wrong semantics for this purpose. A witness atom absent from the cautious consequences might still be derivable in some trace, meaning the corresponding conflict does not actually hold.

Three categories of witness atoms are collected:

```
% True reachability: P appears in some reachable state
true_reachable(P) :- holds(P, T), prop(P).

% Coexistence witness: G1 and G2 hold in a common state
coexist_witness(G1, G2) :- goal(G1), goal(G2), G1 < G2,
    holds(G1, T), holds(G2, T), time(T).

% Sequencing witness: G2 is achievable after G1
g2_after_g1_witness(G1, G2) :- goal(G1), goal(G2), G1 != G2,
    holds(G1, T1), holds(G2, T2), T2 >= T1.
```

Listing 5: Witness atoms for brave reasoning.

Under brave reasoning, `true_reachable(P)` is derived if and only if proposition P appears in at least one state of at least one execution trace. A proposition is truly unreachable only if no execution trace of any length up to the horizon can produce a state containing it.

The `coexist_witness(G1, G2)` atom is derived if some execution trace reaches a state where both G_1 and G_2 hold. The absence of this witness under brave reasoning means that no execution trace, across all possible operator selections, ever produces a state satisfying both goals. This corresponds exactly to the mutex relationship from Definition 22.

The `g2_after_g1_witness(G1, G2)` atom is derived if some execution trace reaches G_1 at time T_1 and G_2 at some $T_2 \geq T_1$. Its absence means that achieving G_1 necessarily prevents subsequent achievement of G_2 in every possible continuation, corresponding to the sequencing conflict from Definition 25.

5.4. Measure Extraction

After the brave reasoning pass completes, the witness atoms are returned as a typed value object. The measure-extraction module then computes the six measure values through pure set operations on those atoms.

Unreachability Profile. The set of reachable propositions is

$$P_{\text{reach}} = \{\, p \mid \texttt{true\_reachable}(p) \,\},$$

and $G_{\text{reach}} = G \cap P_{\text{reach}}$ collects the reachable goal propositions. The unreachability profile is:

$$\mathcal{I}_{\text{UR}}^{\text{scope}}(\Pi, h) = |G \setminus G_{\text{reach}}| \qquad \mathcal{I}_{\text{UR}}^{\text{struct}}(\Pi, h) = |P \setminus P_{\text{reach}}|$$

Goal Mutex Profile. The set of mutex goal pairs is

$$M = \{\, \{g_1, g_2\} \in \tbinom{G_{\text{reach}}}{2} \mid \texttt{coexist\_witness}(g_1, g_2) \text{ not derived} \,\},$$

and $V_{\text{MX}} = \bigcup_{\{g_1,g_2\} \in M} \{g_1, g_2\}$ collects the goals participating in at least one such pair. The mutex profile is:

$$\mathcal{I}_{\text{MX}}^{\text{scope}}(\Pi, h) = |V_{\text{MX}}| \qquad \mathcal{I}_{\text{MX}}^{\text{struct}}(\Pi, h) = |M|$$

Goal Sequencing Profile. The set of ordered sequencing conflict pairs is

$$C = \{\, (g_1, g_2) \in G_{\text{reach}} \times G_{\text{reach}} \mid g_1 \neq g_2, \\ \texttt{g2\_after\_g1\_witness}(g_1, g_2) \text{ not derived} \,\},$$

and $V_{\text{GS}} = \{\, g \mid \exists g' : (g, g') \in C \vee (g', g) \in C \,\}$ collects the goals involved in at least one conflict. The sequencing profile is:

$$\mathcal{I}_{\text{GS}}^{\text{scope}}(\Pi, h) = |V_{\text{GS}}| \qquad \mathcal{I}_{\text{GS}}^{\text{struct}}(\Pi, h) = |C|$$

The resulting six values are packaged into a diagnostic profile data structure that provides automatic category classification according to the taxonomy from Section 4.1. Problem size metadata (goal, proposition, and operator counts) is recorded separately and bundled with the profile and per-phase timing in the result envelope returned to the caller.

5.5. Tooling and Reproducibility

The implementation provides both a library API and a command-line interface (CLI). The library exposes `compute_measures` as the primary entry point, accepting either pre-translated ASP files or PDDL domain and problem files with a horizon parameter h, and returning a result envelope that bundles the diagnostic profile, problem size metadata, and per-phase timing. A timeout-protected variant, `compute_with_timeout`, wraps this entry point for callers that need bounded execution.

Together these directly implement the parameterized measures $\mathcal{I}(\Pi, h)$ from Section 4.4.

A batch processing module iterates over benchmark directories and automatically discovers domain and problem file pairs across three common layouts:

- a single-domain layout with `domain.pddl` paired with `prob*.pddl` or `satprob*.pddl`,
- numbered pairs with `dom01.pddl` and `prob01.pddl`,
- the Eriksson layout with `domain_<name>.pddl` and `<name>.pddl`.

Results are written to CSV format with problem size metadata (goal, proposition, and operator counts), per-phase timing (translation, grounding, solving, extraction, and total), and error reporting.

Both the CLI and the batch runner accept a configurable timeout parameter (in seconds), implemented through the library-level `compute_with_timeout` entry point. When the time limit is exceeded, the problem is recorded with `TIMEOUT` status rather than blocking indefinitely. This is necessary for benchmark domains where grounding complexity causes prohibitive solve times.

Implementing reliable timeouts for ASP solving requires care. Clingo's `--time-limit` option is unavailable in the Python package distribution, and signal-based approaches (e.g., `SIGALRM`) are unreliable because Clingo's C extension can block signal delivery during grounding. The solution runs each computation in a separate child process using Python's `multiprocessing` module with the `spawn` start method, which creates a clean process that can be terminated with `SIGKILL` after the timeout, guaranteeing termination regardless of the solver's internal state.

For reproducibility, the implementation is containerized with Docker, bundling Python 3.12, Clingo 5.8.0, and *plasp* 3 in a consistent environment. The Dockerfile and build instructions are included in the repository.

5.6. Validation

Correctness is verified through an automated test suite comprising 80 tests across eight categories, with the per-phase decomposition of the pipeline mirrored by dedicated test files for the translation, brave-reasoning, and extraction seams. The full suite executes in under one second, enabling rapid regression testing during development.

Scenario Tests. Eleven test scenarios are tested against their theoretically derived expected profiles. Table 8 shows the computed profiles alongside the expected values. All eleven scenarios produce exact matches, including the *Negative Precondition* scenario that exercises the `neg_precond` encoding path.

Category	Scenario	Expected	Computed
P1 Unreachability	Locked Door	$(1, 3, 0, 0, 0, 0)$	$(1, 3, 0, 0, 0, 0)$
	Bank Vault	$(1, 7, 0, 0, 0, 0)$	$(1, 7, 0, 0, 0, 0)$
P2 Mutex	Light Switch	$(0, 0, 2, 1, 0, 0)$	$(0, 0, 2, 1, 0, 0)$
	Traffic Light	$(0, 0, 3, 3, 0, 0)$	$(0, 0, 3, 3, 0, 0)$
P3 Sequencing	Trust and Travel	$(0, 0, 2, 1, 2, 1)$	$(0, 0, 2, 1, 2, 1)$
	Rival Alliances	$(0, 0, 2, 1, 2, 2)$	$(0, 0, 2, 1, 2, 2)$
Edge Cases	Coexisting Goals	$(0, 0, 0, 0, 0, 0)$	$(0, 0, 0, 0, 0, 0)$
	Single Goal	$(0, 0, 0, 0, 0, 0)$	$(0, 0, 0, 0, 0, 0)$
	Empty Goals	$(0, 0, 0, 0, 0, 0)$	$(0, 0, 0, 0, 0, 0)$
	Delete Relaxation	$(1, 1, 0, 0, 0, 0)$	$(1, 1, 0, 0, 0, 0)$
	Negative Precondition	$(1, 1, 0, 0, 0, 0)$	$(1, 1, 0, 0, 0, 0)$

Table 8: Computed profiles for test scenarios.

Hierarchy Tests. Four tests verify the measure relationships established in Section 4.5.2. These tests confirm that unreachable goals do not participate in mutex or sequencing conflicts, that sequencing conflicts imply mutex, that mutex without sequencing indicates reversible conflicts, and that delete effects correctly prevent false reachability claims.

Dataclass Contract Tests. Seventeen tests verify the publicly accessible dataclasses (`MeasureProfile`, `ProblemSize`, `TimingProfile`, `MeasureResult`), confirming category classification, the stable CSV field order, frozen-dataclass semantics, and the human-readable summary that crosses profile, size, and timing.

Extraction Tests. Eleven tests exercise the pure set-algebra extraction module on synthetic atom sets, covering unreachability counting, the directional treatment of sequencing witnesses, the exclusion of unreachable goals from mutex and sequencing analyses, and the derivation of problem size cardinalities.

Brave Reasoning Tests. Two tests confirm that the brave-reasoning runner returns a populated outcome on a known scenario and surfaces unsatisfiable programs as a clear error.

Pipeline API Tests. Five tests on `compute_measures` verify that the pipeline returns a fully populated result envelope, raises appropriate errors for missing files and unsatisfiable programs, captures horizon sensitivity, and emits a timing profile with positive grounding and solving times for `.lp` input.

Execution and CSV Tests. Seven tests verify the timeout-protected `compute_with_timeout` entry point and the `ExecutionResult` discriminator, covering the `OK`, `TIMEOUT`, and `ERROR` statuses, CSV row composition with stable field order, and the collapsing of multi-line error messages into a single CSV cell.

PDDL Pipeline Tests. Twenty-three tests cover the PDDL ingestion path. Six tests confirm that the *plasp* translation, the bridge encoding, and the context-managed translated-problem handle produce correct measure profiles end-to-end and surface translation failures cleanly. Eleven tests verify the cost-stripping splicer and the context-manager cleanup it shares with the translator. Six tests verify that benchmark discovery correctly identifies domain/problem pairs across the three supported layouts (single-domain, numbered, and Eriksson) and respects the skip list of known-incompatible domains.

With the pipeline validated on controlled scenarios, Section 6 applies it to standard unsolvable planning benchmarks to assess the practical diagnostic value of the measures.

6. Evaluation

This section evaluates the implemented measures on unsolvable planning benchmarks from the IPC 2016 Unsolvability Competition. The evaluation addresses Research Question 3, asking whether the adapted measures provide practical diagnostic value for understanding and analyzing unsolvable planning problems. Section 6.1 describes the experimental setup and dataset. Section 6.2 presents the benchmark results across seven compatible domains. Section 6.3 examines individual problems that illustrate the diagnostic capabilities and limitations of the measures. Section 6.4 analyzes the sensitivity of results to the horizon parameter, revealing a soundness-coverage tradeoff that constitutes a key methodological finding. Section 6.5 discusses computational performance and scalability. Section 6.6 interprets the findings and addresses limitations.

6.1. Evaluation Methodology

The evaluation methodology covers the choice of benchmark suites and the experimental configuration under which all reported measurements are collected. The dataset selection determines which categories of unsolvability are exercised by the experiments, while the experimental setup fixes the horizon, time budget, and hardware against which the scalability claims in Section 6.5 must be read.

6.1.1. Dataset

The evaluation uses two benchmark suites. The primary dataset is the IPC 2016 Unsolvability Competition benchmark suite by Muise and Lipovetzky [32], which

contains 15 domains of unsolvable planning problems specifically designed for evaluating unsolvability detection techniques. Each domain includes between 15 and 25 unsolvable problem instances of increasing size, and several domains also include solvable instances (`satprob` files) for validation purposes. The secondary dataset is the unsolvable benchmark collection by Eriksson [13], which extends the IPC 2016 domains and adds new domains including *3unsat*, a set of unsatisfiable 3-SAT instances encoded as planning problems.

Of the 15 IPC 2016 domains, 6 are compatible with the implementation pipeline. One additional domain from the Eriksson benchmarks, *3unsat*, is also compatible. Table 9 summarizes the compatibility status. Two domains (*bag-barman* and *tetris*) use the `:equality` requirement, which *plasp* does not support, as noted in Section 5.2. Seven domains produce grounding programs that exceed the 120-second time limit: *bag-gripper*, *bag-transport*, *over-nomystery*, *over-rovers*, *over-tpp*, *sliding-tiles*, and *pegsol*. The compatible domains span a range of unsolvability types: *diagnosis* exhibits unreachability, mutex, and sequencing conflicts; *bottleneck* and *chessboard-pebbling* exhibit pure mutex and sequencing conflicts without unreachability; *cave-diving*, *document-transfer*, and *pegsol-row5* primarily exhibit unreachability; and *3unsat* encodes combinatorial unsatisfiability outside the scope of the proposed measures.

Domain	**Status**	**Problems**	**Reason for Incompatibility**
diagnosis	Compatible	20	
bottleneck	Compatible	25	
cave-diving	Compatible	25	
document-transfer	Compatible	20	
pegsol-row5	Compatible	15	
chessboard-pebbling	Compatible	23	
3unsat	Compatible	30	
bag-barman	Incompatible		`:equality` unsupported
tetris	Incompatible		`:equality` unsupported
bag-gripper	Incompatible		Grounding timeout
bag-transport	Incompatible		Grounding timeout
over-nomystery	Incompatible		Grounding timeout
over-rovers	Incompatible		Grounding timeout
over-tpp	Incompatible		Grounding timeout
sliding-tiles	Incompatible		Grounding timeout
pegsol	Incompatible		Grounding timeout

Table 9: Domain compatibility with the implementation pipeline.

6.1.2. Experimental Setup

All experiments run inside a Docker container with Python 3.12, Clingo 5.8.0, and *plasp* 3.1.1 on a host with an AMD Ryzen 5 5600X CPU (6 cores, 12 threads, 4.65 GHz, 32 MiB L3) and 32 GiB of RAM. Clingo is invoked single-threaded through its Python API. The horizon parameter is set to $h = 20$ for all primary runs, and each problem has a time limit of 120 seconds. The batch runner (Section 5.5) processes all domain/problem pairs in each compatible domain, recording the full diagnostic profile, category classification, problem size metadata (goal, proposition, and operator counts), per-phase computation times (translation, grounding, solving, extraction), and status (OK, TIMEOUT, or ERROR) for each instance. Additional runs at $h = 10$ and $h = 30$ are performed on selected instances for the horizon sensitivity analysis in Section 6.4.

6.2. Benchmark Results

This section reports the measured diagnostic profiles for the 158 unsolvable problems across the seven compatible domains. The presentation proceeds from aggregate coverage and category classification distribution to a detailed per-instance table for the most informative domain (*diagnosis*) and summary observations for the remaining six. Per-instance case studies follow in Section 6.3.

6.2.1. Overall Coverage

Of the 158 unsolvable problems across the seven compatible domains, 68 (43%) complete within the 120-second time limit. Table 10 shows the per-domain breakdown. Coverage varies substantially: *diagnosis* achieves 90% coverage, while *chessboard-pebbling* covers only 13%. The primary limiting factor is grounding complexity, as discussed in Section 6.5. Two solvable instances also complete successfully: *cave-diving* `satprob01` and *document-transfer* `satprob01`, both correctly classified as consistent.

Domain	Total	OK	Timeout	Coverage
diagnosis	20	18	2	90%
bottleneck	25	12	13	48%
cave-diving	25	9	16	36%
document-transfer	20	3	17	15%
pegsol-row5	15	3	12	20%
chessboard-pebbling	23	3	20	13%
3unsat	30	20	10	67%
Total	**158**	**68**	**90**	**43%**

Table 10: Per-domain coverage at $h = 20$ with 120-second time limit.

The per-instance results for each domain are tabulated in Appendix A.

6.2.2. Category Distribution

Figure 4 shows the distribution of category classifications among the 68 successfully computed unsolvable instances. Category 2a (unreachability) is the most common classification, appearing in 25 instances across four domains. Category 2c-sequencing appears in 13 instances across two domains. No instance exhibits a pure 2c-mutex classification (mutex without sequencing). 30 instances are classified as consistent: 10 from the IPC 2016 domains, where Section 6.4 shows these are horizon-dependent, and 20 from *3unsat*, where the consistent classification is horizon-independent (Section 6.3.6).

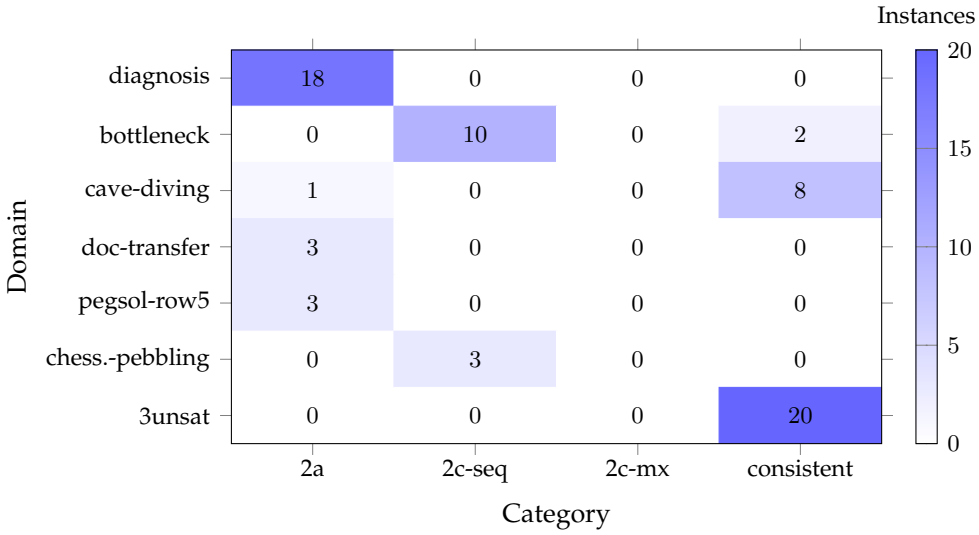

Figure 4: Category classifications per domain across unsolvable instances.

The absence of pure 2c-mutex classifications is consistent with the theoretical relationship from Proposition 15. Every instance with $\mathcal{I}_{\text{GS}}^{\text{scope}}(\Pi, 20) > 0$ also has $\mathcal{I}_{\text{MX}}^{\text{scope}}(\Pi, 20) > 0$, which is consistent with the proposition that sequencing conflicts imply mutex. The unit test scenarios in Section 5.6 confirm that pure mutex cases are correctly handled.

The 30 consistent classifications fall into two groups. Ten IPC 2016 instances exhibit horizon-dependent behavior (Section 6.4), while the 20 *3unsat* instances are horizon-independent (Section 6.3.6).

6.2.3. Diagnosis Domain

The *diagnosis* domain provides the richest results, with the highest coverage (90%), and it contains the only benchmark instances exhibiting all three measure types. Table 11 shows the full results.

	Size			UR		MX		GS		
Problem	$\|G\|$	$\|P\|$	$\|O\|$	$\mathcal{I}^s$	$\mathcal{I}^t$	$\mathcal{I}^s$	$\mathcal{I}^t$	$\mathcal{I}^s$	$\mathcal{I}^t$	**Time**
prob01	16	190	216	9	16	0	0	0	0	0.5s
prob02	16	190	216	9	16	0	0	0	0	0.5s
prob03	16	155	125	10	20	0	0	0	0	0.2s
prob04	38	389	568	31	59	0	0	0	0	1.6s
prob05	56	542	957	49	97	0	0	0	0	2.9s
prob06	7	49	25	1	1	0	0	0	0	0.04s
prob07	22	154	81	12	31	0	0	0	0	0.1s
prob08	24	170	96	14	41	0	0	0	0	0.2s
prob09	24	241	156	14	33	0	0	0	0	43.2s
prob10	24	187	99	14	25	2	1	2	2	0.3s
prob11	22	204	139	14	33	0	0	0	0	2.8s
prob12	24	199	99	15	26	0	0	0	0	0.5s
prob13	36	399	232	27	67	0	0	0	0	0.9s
prob14	28	314	152	19	38	0	0	0	0	1.4s
prob15	24	240	155	14	35	0	0	0	0	0.5s
prob16	36	457	257	20	32	8	28	8	56	2.5s
prob17	24	214	120	15	27	0	0	0	0	0.5s
prob18					TIMEOUT					
prob19	24	214	120	15	27	0	0	0	0	0.5s
prob20					TIMEOUT					

Table 11: Diagnostic profiles for *diagnosis* domain at $h = 20$.

All 18 successfully computed instances are classified as Category 2a, indicating that unreachable goals are present in every case. Unreachability is the dominant source of inconsistency, with $\mathcal{I}_{\text{UR}}^{\text{scope}}$ ranging from 1 in prob06 to 49 in prob05, representing between 14% and 88% of goals. Among the 18 instances, only prob10 and prob16 exhibit non-zero mutex and sequencing measures, making them the structurally richest problems in the benchmark suite. The scope-structure bounds from Proposition 16 hold across all instances. For example, prob16 has $\mathcal{I}_{\text{MX}}^{\text{struct}}(\text{prob16}, 20) = 28 = \binom{8}{2}$, achieving the maximum for 8 mutex goals and indicating that every pair of mutex goals is in conflict with every other.

6.2.4. Other Domains

Beyond *diagnosis*, five further benchmark domains yield instances that complete within the time limit. Each is discussed in turn below, reporting the completion rate, the dominant conflict category, and the structural pattern that characterizes it.

Bottleneck. Twelve of 25 instances complete, all exhibiting zero unreachability. Ten are classified as 2c-sequencing and two as consistent. Table 12 shows the full results. The instances are grouped by grid dimension, with 4, 5, or 6 goals respectively. Within each group, $\mathcal{I}_{\text{MX}}^{\text{scope}}$ decreases from the full goal count to 2 and then to 0 as the bottleneck position shifts. In the 4-goal group, $\mathcal{I}_{\text{MX}}^{\text{scope}}$ drops from 4 in prob01 to 2 in prob03. In the 5-goal group, it drops from 5 in prob04 to 2 in prob07. In the 6-goal group, it drops from 6 in prob08 to 0 in prob11 and prob12. The two consistent instances, prob11 and prob12, have no actual pairwise structural conflicts at $h = 20$. Section 6.4 shows that a shorter horizon of $h = 10$ produces spurious conflict detections on these instances.

Problem	$\lvert G\rvert$	$\lvert P\rvert$	$\lvert O\rvert$	$\mathcal{I}_{\text{MX}}^{s}$	$\mathcal{I}_{\text{MX}}^{t}$	$\mathcal{I}_{\text{GS}}^{s}$	$\mathcal{I}_{\text{GS}}^{t}$	**Cat.**	**Time**
prob01	4	629	4,913	4	6	4	12	2c-seq	2.1s
prob02	4	702	5,832	3	3	3	6	2c-seq	2.3s
prob03	4	779	6,859	2	1	2	2	2c-seq	2.8s
prob04	5	1,430	17,576	5	10	5	20	2c-seq	10.4s
prob05	5	1,539	19,683	4	6	4	12	2c-seq	11.4s
prob06	5	1,652	21,952	3	3	3	6	2c-seq	11.6s
prob07	5	1,769	24,389	2	1	2	2	2c-seq	14.5s
prob08	6	2,849	50,653	6	15	6	30	2c-seq	32.1s
prob09	6	3,002	54,872	4	4	4	8	2c-seq	46.7s
prob10	6	3,159	59,319	2	1	2	2	2c-seq	51.9s
prob11	6	3,320	64,000	0	0	0	0	cons.	62.1s
prob12	6	3,485	68,921	0	0	0	0	cons.	67.8s

Table 12: Diagnostic profiles for *bottleneck* domain at $h = 20$.

Cave-diving. Nine of 25 unsolvable instances complete, of which eight are classified as consistent and one (prob05) as Category 2a with a single unreachable goal. The consistent classifications indicate that no pairwise structural conflicts exist among the goals at $h = 20$. The unsolvability of these instances arises from properties outside the scope of pairwise analysis. Section 6.4 shows that a shorter horizon of $h = 10$ produces spurious conflict detections on these instances, but the same spurious detections also appear on the solvable `satprob01`, confirming that they are artifacts of insufficient exploration depth.

Document-transfer. Only 3 of 20 unsolvable instances complete due to the large grounding sizes. All three are classified as Category 2a with heavy unreachability: prob01 and prob03 have 19 out of 19 goals unreachable, indicating that no goal proposition is reachable in these instances, while prob06 has 1 of 4 goals unreachable. The domain also contains 9 instances with unknown solvability status (`unknownprob` files), all of which exceed the time limit.

Pegsol-row5. Three of 15 instances complete, all classified as Category 2a with exactly one unreachable goal. The domain exhibits extreme exponential growth in problem size: prob01 has 228 propositions and 216 operators, prob02 has 2,772 and 2,744, and prob03 has 13,872 and 13,824, creating a sharp computational wall after prob03.

Chessboard-pebbling. Three of 23 instances complete, all classified as 2c-sequencing with an identical conflict structure of $\mathcal{I}_{\text{MX}}^{\text{scope}}(\Pi, 20) = 3$, $\mathcal{I}_{\text{MX}}^{\text{struct}}(\Pi, 20) = 2$, $\mathcal{I}_{\text{GS}}^{\text{scope}}(\Pi, 20) = 3$, $\mathcal{I}_{\text{GS}}^{\text{struct}}(\Pi, 20) = 2$. All three goals participate in conflicts regardless of grid size, with operator counts growing from 15,625 in prob03 to 46,656 in prob04 and 117,649 in prob05. This structural invariance indicates that the conflict topology is determined by the goal specification, not by problem scale.

6.3. Case Studies

This section examines six benchmark instances in detail. The selected cases cover an instance where all three measures are non-zero, the densest conflict structure observed in the suite, a solve-phase outlier, a domain whose conflict structure remains invariant across problem scale, an unreachability-only case in a resource domain, and an inherent blind spot of pairwise analysis.

6.3.1. Diagnosis prob10: All Three Measures

Instance prob10 is a benchmark problem where all three measures are non-zero, with profile $(14, 25, 2, 1, 2, 2)$. Of the 24 goals, 14 are unreachable, leaving 10 achievable goals ($\mathcal{I}_{\text{UR}}^{\text{scope}}(\text{prob10}, 20) = 14$). Among these, 2 goals are in a single mutex pair ($\mathcal{I}_{\text{MX}}^{\text{scope}}(\text{prob10}, 20) = 2$, $\mathcal{I}_{\text{MX}}^{\text{struct}}(\text{prob10}, 20) = 1$), and the same 2 goals form 2 ordered sequencing conflicts ($\mathcal{I}_{\text{GS}}^{\text{scope}}(\text{prob10}, 20) = 2$, $\mathcal{I}_{\text{GS}}^{\text{struct}}(\text{prob10}, 20) = 2$). The sequencing structure count of 2 for 2 goals indicates both orderings (g_1, g_2) and (g_2, g_1) are conflicts, meaning achieving either goal permanently prevents the other. This is consistent with an irreversible exclusion in Category 2c.

This instance demonstrates the complementary diagnostic value of the measures. The unreachability profile identifies 14 goals that cannot be satisfied under any execution trace. The mutex profile identifies 2 additional goals that are individually achievable but cannot coexist in any reachable state. The sequencing profile further specifies that this mutex is caused by irreversible effects rather than by reversible state constraints. A planning engineer could use this decomposition to distinguish between goals requiring new operators (unreachable) and goals requiring operator modification (mutex/sequencing).

6.3.2. Diagnosis prob16: Dense Conflict Structure

Instance prob16 exhibits the densest conflict structure in the benchmark suite, with profile $(20, 32, 8, 28, 8, 56)$. Of 36 goals, 20 are unreachable. Among the 16 achievable

goals, 8 participate in mutex relationships forming 28 unordered pairs, and the same 8 goals produce 56 ordered sequencing conflicts. The mutex structure count of 28 equals $\binom{8}{2}$, meaning every pair of the 8 conflicting goals is mutex. The sequencing count of 56 equals 8×7, meaning every ordered pair is a sequencing conflict. This indicates a clique structure where the 8 goals form a complete conflict graph and achieving any one permanently blocks all others.

6.3.3. Diagnosis prob09: Solve-Phase Outlier

Instance prob09 has a moderate problem size of 24 goals, 241 propositions, and 156 operators, comparable to prob08 (24 goals, 170 propositions, 96 operators), which completes in 0.20 seconds. However, prob09 requires 43.2 seconds, making it the second-slowest completing instance across all domains. The profile $(14, 33, 0, 0, 0, 0)$ shows only unreachability.

The per-phase runtime breakdown reveals the cause. Grounding completes in 0.11 seconds, comparable to prob08 at 0.04 seconds and well within the range of other *diagnosis* instances. The solving phase, however, takes 43.0 seconds, accounting for 99.7% of total time. The brave reasoning search is combinatorially expensive for this particular problem structure, likely due to the interaction pattern between the 156 operators and the reachability witnesses across 20 time steps.

This case demonstrates that operator count is not a reliable predictor of computation time. The ground program size, determined by $|O| \times h$, governs grounding cost, but the combinatorial structure of the specific operator interactions governs solving cost. Two instances with similar $|O|$ can differ by orders of magnitude in solve time.

6.3.4. Chessboard-pebbling: Structural Invariance

The three successfully computed instances of *chessboard-pebbling* (prob03, prob04, prob05) produce identical conflict profiles despite substantially different problem sizes. All three have 3 goals, all 3 participate in mutex and sequencing conflicts, and the structure counts are constant ($\mathcal{I}^{\text{struct}}_{\text{MX}}(\Pi, 20) = 2$, $\mathcal{I}^{\text{struct}}_{\text{GS}}(\Pi, 20) = 2$). The grid sizes grow from 1,300 to 4,900 propositions and from 15,625 to 117,649 operators, yet the conflict structure remains invariant. This confirms that the measures capture structural properties of the goal specification rather than artifacts of problem scale.

6.3.5. Cave-diving prob05: Unreachability in a Resource Domain

Among the nine completed *cave-diving* instances, prob05 is the only one classified as Category 2a, with profile $(1, 49, 0, 0, 0, 0)$. One of the three goals is unreachable, in contrast to the other eight completed instances where all goals are reachable at $h = 20$. This suggests that prob05 contains a missing producer (Category 2a), while the other instances have no detectable pairwise structural conflicts, placing their unsolvability outside the scope of the proposed measures.

6.3.6. 3unsat: Inherent Measure Blind Spot

The *3unsat* domain from the Eriksson benchmark collection [13] encodes unsatisfiable 3-SAT instances as planning problems. Each problem has a set of Boolean variables as operators (assigning true or false) and a set of clauses as goals (each clause is "solved" when at least one of its literals is satisfied). All 20 completed instances return the profile $(0, 0, 0, 0, 0, 0)$ with $\mathcal{I}_{\text{UR}}^{\text{struct}}(\Pi, 20) = 0$. Not a single proposition is unreachable, and no mutex or sequencing conflicts are detected at any horizon.

The *3unsat* consistent classification is fundamentally different from those observed in cave-diving and bottleneck. Testing at $h = 3$ (the minimum horizon needed for a single variable assignment) confirms that the profile remains $(0, 0, 0, 0, 0, 0)$. Each goal clause is individually achievable (by assigning the appropriate variable values), and for every pair of clauses, some reachable state satisfies both (by choosing compatible assignments). The unsolvability arises because no single assignment satisfies *all* clauses, a combinatorial constraint that pairwise analysis cannot detect.

The *3unsat* blind spot represents an inherent limitation of the proposed measures, corresponding to the higher-order goal analysis discussed in Section 7. The measures detect pairwise conflicts. For problems with $|G| \leq 2$, zero measures imply solvability. For larger goal sets, pairwise achievability does not guarantee that the full goal set is achievable. The *3unsat* domain provides the first empirical example of this limitation on benchmark data, complementing the theoretical observation in the future work discussion.

6.4. Horizon Sensitivity Analysis

The horizon parameter h bounds the number of time steps explored during brave reasoning. This section presents evidence that the choice of horizon significantly affects the computed profiles, revealing a soundness-coverage tradeoff inherent to horizon-bounded brave reasoning.

6.4.1. Mechanism

Under the horizon-bounded brave reasoning approach, longer horizons allow more propositions to become reachable through longer operator chains. Since the encoding explores nondeterministic execution traces up to the given horizon bound, the interaction between horizon length and conflict detection is non-trivial. The formal properties established in Proposition 2 predict three effects, all of which are observed empirically:

1. **Monotonically decreasing unreachability**: As h increases, $S_{\text{reach}}^{h} \subseteq S_{\text{reach}}^{h+1}$ by Proposition 2, so $\mathcal{I}_{\text{UR}}^{\text{scope}}(\Pi, h)$ is monotonically non-increasing in h.
2. **Non-monotonic mutex and sequencing**: As noted in Section 4.4.2, $\mathcal{I}_{\text{MX}}^{\text{scope}}(\Pi, h)$ and $\mathcal{I}_{\text{GS}}^{\text{scope}}(\Pi, h)$ are not monotonic in h, because increasing h both makes new

goals reachable, which may introduce apparent conflicts, and provides longer witness traces, which may resolve previously reported conflicts.

3. **False positives at short horizons**: An insufficient horizon may fail to find witnesses that exist at longer traces, reporting conflicts that do not represent actual structural limitations. These are correct values of $\mathcal{I}(\Pi, h)$ for the given h but do not reflect the converged values $\mathcal{I}(\Pi, h^*)$.

Monotonic unreachability, non-monotonic mutex and sequencing, and short horizon false positives together describe the soundness side of the tradeoff. The coverage side reflects encoding cost. Each additional time step enlarges the grounded program, so longer horizons increase grounding and solving time within the time limit and reduce the number of instances that complete. The choice of h therefore balances soundness against coverage rather than admitting a single optimum.

6.4.2. Evidence: Diagnosis prob10

Table 13 shows the profile of *diagnosis* prob10 at three horizon values. At $h = 10$, $\mathcal{I}^{\text{scope}}_{\text{UR}}(\text{prob10}, 10) = 19$ and no mutex or sequencing conflicts are detected because the potentially conflicting goals are not yet reachable. At $h = 20$, five additional goals become reachable, two of which appear in mutex and sequencing conflict with $\mathcal{I}^{\text{scope}}_{\text{MX}}(\text{prob10}, 20) = 2$ and $\mathcal{I}^{\text{scope}}_{\text{GS}}(\text{prob10}, 20) = 2$. At $h = 30$, three more goals become reachable and the mutex and sequencing measures return to zero with $\mathcal{I}^{\text{scope}}_{\text{MX}}(\text{prob10}, 30) = 0$. This demonstrates the non-monotonicity predicted in Section 4.4.2. The conflicts reported at $h = 20$ are horizon artifacts rather than actual structural limitations, as longer traces provide witnesses for goal co-existence. The prob10 case illustrates that the horizon sensitivity is not limited to the distinction between "consistent" and "conflicting" classifications. Even non-zero conflict detections can be artifacts of an intermediate horizon that has not yet converged.

h	$\mathcal{I}^s_{\text{UR}}$	$\mathcal{I}^t_{\text{UR}}$	$\mathcal{I}^s_{\text{MX}}$	$\mathcal{I}^t_{\text{MX}}$	$\mathcal{I}^s_{\text{GS}}$	$\mathcal{I}^t_{\text{GS}}$
10	19	39	0	0	0	0
20	14	25	2	1	2	2
30	11	17	0	0	0	0

Table 13: Horizon sensitivity for *diagnosis* prob10.

6.4.3. Evidence: Cave-diving and Bottleneck

The ten instances classified as consistent at $h = 20$ from *cave-diving* and *bottleneck* are all spuriously classified as 2c-sequencing when the horizon is reduced to $h = 10$. Table 14 shows the *cave-diving* results. At $h = 10$, every instance, including the solvable `satprob01`, shows all goals in apparent mutex and sequencing conflict. At $h = 20$,

the horizon is sufficient for brave reasoning traces to establish goal co-existence, and all instances are correctly classified as consistent. The solvable `satprob01` exhibiting the same spurious conflicts as the unsolvable instances confirms that these detections are artifacts of insufficient exploration depth, not actual structural pairwise conflicts. This false positive also provides empirical justification for the choice of $h = 20$ as the default horizon, as no false positives are observed among the solvable instances that complete at this horizon.

Problem	**Solvable?**	$\|G\|$	$\mathcal{I}^s_{\text{MX}}$	$\mathcal{I}^t_{\text{MX}}$	$\mathcal{I}^s_{\text{GS}}$	$\mathcal{I}^t_{\text{GS}}$	**Category**
At $h = 10$:							
prob01	No	3	3	2	3	4	2c-sequencing
prob02	No	3	3	2	3	4	2c-sequencing
prob03	No	3	3	2	3	4	2c-sequencing
prob07	No	6	6	9	6	18	2c-sequencing
prob08	No	6	6	9	6	18	2c-sequencing
prob12	No	5	5	4	5	8	2c-sequencing
prob13	No	5	5	4	5	8	2c-sequencing
prob14	No	5	5	4	5	8	2c-sequencing
satprob01	Yes	3	3	2	3	4	2c-sequencing
At $h = 20$: all of the above classified as consistent							

Table 14: Cave-diving at $h = 10$ versus $h = 20$.

6.4.4. Interpretation

The horizon sensitivity analysis yields three conclusions. First, each computed profile is a correct value of $\mathcal{I}(\Pi, h)$ for the given horizon h; the classification "consistent" at a given horizon means that no pairwise structural conflicts are detectable at that exploration depth, not that the converged measures $\mathcal{I}(\Pi, h^*)$ are zero. Second, $h = 20$ represents a practical balance. It avoids false positives on the available solvable instances while detecting actual unreachability in 25 of 68 unsolvable instances and mutex/sequencing conflicts in 13 additional instances. Third, the ten consistent IPC 2016 classifications at $h = 20$ are not evidence of missed pairwise conflicts. A shorter horizon of $h = 10$ produces apparent conflict detections on these instances, but the identical detections on the solvable `satprob01` demonstrate that these are artifacts of insufficient exploration depth rather than actual structural limitations. The unsolvability of these instances, like that of the 20 horizon-independent *3unsat* instances (Section 6.3.6), lies outside the scope of pairwise analysis.

6.4.5. Practical Mitigation

The non-monotonic behavior of mutex and sequencing measures suggests three practical strategies for filtering horizon-induced false positives.

Multi-horizon comparison. Running the analysis at two or more horizon values and comparing profiles is the most direct filtering strategy. If a conflict detected at horizon h disappears at $h + \Delta$, the longer horizon has found a witness trace that resolves the apparent conflict. The prob10 case (Table 13) illustrates this. The mutex and sequencing conflicts at $h = 20$ vanish at $h = 30$, identifying them as horizon artifacts. The computational cost scales linearly with the number of horizon values tested.

Unreachability convergence check. Since $\mathcal{I}_{\text{UR}}^{\text{scope}}(\Pi, h)$ is monotonically non-increasing in h by Proposition 2, a decrease in unreachability between successive horizons indicates that the reachable state space has not yet converged. In this case, mutex and sequencing values at the shorter horizon are unreliable, as newly reachable goals at longer horizons may resolve apparent conflicts. Conversely, if unreachability is stable across two successive horizons, the state space is likely near convergence, lending confidence to the mutex and sequencing values.

Solvable instance calibration. When solvable instances from the same domain are available, running the analysis on these instances provides a direct false positive check. Any conflict detected on a solvable instance is necessarily a false positive, as demonstrated by *cave-diving* `satprob01` at $h = 10$ (Table 14). If solvable and unsolvable instances produce identical conflict patterns at a given horizon, the detections are likely artifacts rather than actual structural conflicts.

6.5. Performance and Scalability

This section asks where the pipeline spends its time and how that cost scales with problem size. The analysis decomposes each completed run into PDDL translation, Clingo grounding, Clingo solving, and measure extraction, contrasts the per-phase contributions across domains, and re-runs a probe set of timed-out instances under an extended budget to identify which phase causes the 90 timeouts. The result is a two-pattern picture, with a solve-dominated regime that determines runtime for the 70 completed instances and a grounding-dominated regime that bounds scalability on the operator-heavy domains.

6.5.1. Computation Time

The implementation records per-phase runtime per computation, separating PDDL translation, Clingo grounding, Clingo solving (brave reasoning), and Python measure extraction. Among the 70 successfully computed instances (68 unsolvable plus 2 solvable), total computation times range from 0.04 seconds for *diagnosis* prob06, with 49 propositions and 25 operators, to 84.5 seconds for *chessboard-pebbling* prob05, with 4,900 propositions and 117,649 operators. Table 15 summarizes the per-phase runtime distribution.

Translation and extraction are negligible across all domains. Translation completes in under 150 milliseconds and extraction in under 1 millisecond for every instance. The computational cost is therefore entirely determined by the grounding and solving phases, whose relative contributions per domain are shown in Figure 7 and discussed in Section 6.5.2.

Domain	n	**Grounding (s)**			**Solving (s)**		
		Min	Med	Max	Min	Med	Max
diagnosis	18	0.01	0.05	0.35	0.02	0.39	43.0
bottleneck	12	0.83	4.28	13.99	1.26	8.66	53.8
cave-diving	10	0.10	1.27	4.95	0.43	4.23	35.0
document-transfer	4	0.12	1.50	3.51	9.70	18.75	47.3
pegsol-row5	3	0.02	0.37	2.17	0.004	0.10	1.90
chessboard-pebbling	3	3.02	8.68	22.79	7.34	35.8	61.7
3unsat	20	0.06	0.51	1.50	0.20	2.32	39.2

Table 15: Per-phase runtime breakdown for successfully computed instances.

Figure 5 plots total computation time against operator count $|O|$ on a log-log scale across all successfully computed instances. The points follow a near-linear log-log band, consistent with polynomial growth of runtime in $|O|$, which matches the $O(|O| \times h)$ scaling of the timestep-indexed encoding in Section 5.3. The vertical spread within fixed-$|O|$ clusters, most visible for *3unsat* where each problem group fixes $|O|$ exactly, foreshadows the instance-specific solve variance analyzed in Section 6.5.2. The 120-second time limit acts as a ceiling that prevents larger instances from completing.

6.5.2. Phase Analysis

The per-phase breakdown reveals that solving, not grounding, is the dominant cost for instances that complete within the time limit. Figure 6 plots grounding time against solving time for all 70 completing instances. Nearly all points fall above the diagonal, indicating that the solving phase takes longer than grounding. Figure 7 summarizes the median proportions per domain.

Solving accounts for the majority of computation time in six of seven domains (Table 15). The solve-to-ground ratio increases with problem size. In *bottleneck*, the ratio grows from 1.5 for prob01 (4,913 operators) to 3.8 for prob12 (68,921 operators). The most extreme cases occur in *document-transfer*, where prob06 spends 0.12 seconds grounding but 18.8 seconds solving (ratio 154), and the solvable `satprob01` spends 0.25 seconds grounding but 47.3 seconds solving (ratio 191). In these instances, the ground program is small enough to instantiate quickly, but the brave reasoning search across 20 time steps with delete effects creates a combinatorially large search space.

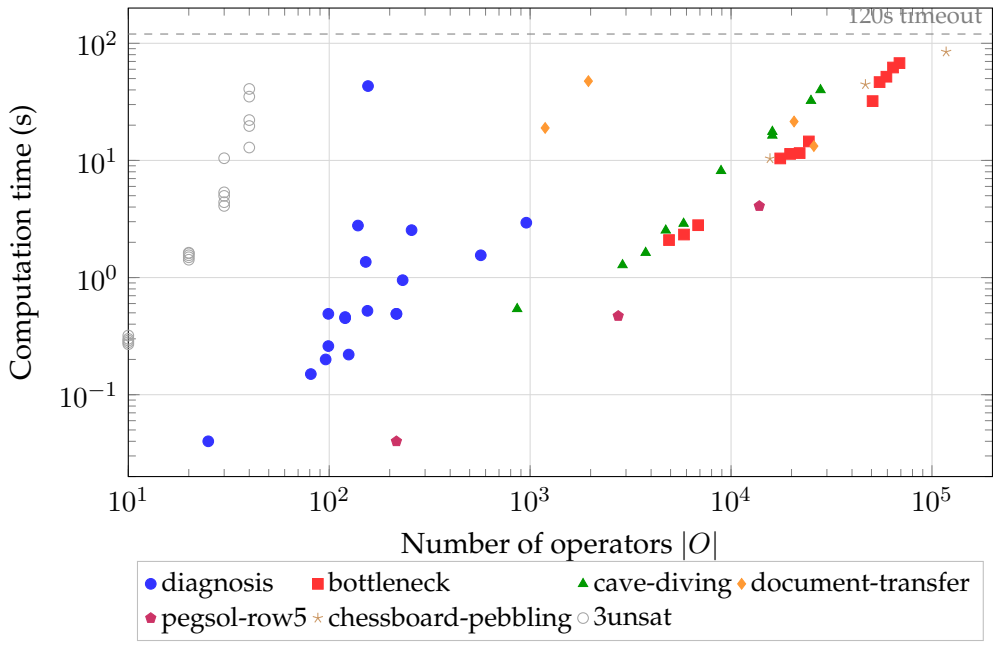

Figure 5: Computation time vs. operator count.

The sole exception is *pegsol-row5*, where grounding dominates. Prob01 spends 0.019 seconds grounding and 0.004 seconds solving; prob02 spends 0.37 seconds grounding and 0.10 seconds solving. By prob03, the phases are roughly equal (2.17 seconds grounding, 1.90 seconds solving). This domain's grid structure produces a large number of ground rules per operator due to the interaction patterns between positions. The contrast with *document-transfer*, where the ground program is tiny but the brave reasoning search explodes, illustrates that grounding cost and solving cost respond to orthogonal properties of the problem. Grounding scales with the static encoding size $|O| \times h$, while solving scales with the combinatorial difficulty of the brave reasoning search across delete effects.

The *diagnosis* prob09 outlier, previously noted at 43.2 seconds total despite only 241 propositions and 156 operators, is now explained by the phase breakdown: grounding completes in 0.11 seconds, comparable to similarly sized instances, while the solving phase takes 43.0 seconds. The brave reasoning search is combinatorially expensive for this particular problem structure despite a moderate ground program. This demonstrates that operator count alone is not a reliable predictor of computation time.

Two Bottleneck Patterns. The per-phase data suggests two distinct scalability patterns. For the 70 instances that complete within the time limit, the solving phase dominates and determines computation time. For the 90 instances that exceed the time limit, amounting to 57% of the total, the phase responsible is not directly ob-

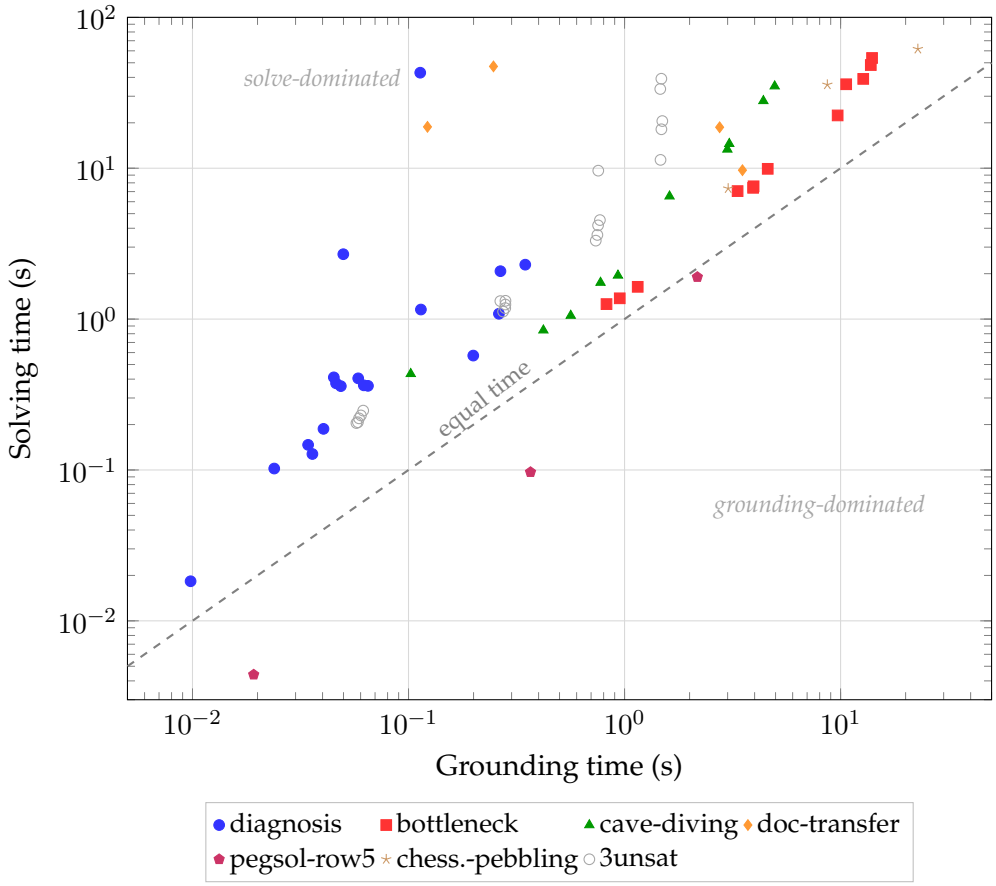

Figure 6: Grounding versus solving time across 70 completed instances.

servable from the default run. To probe this second pattern empirically, the two timed-out instances per domain with the smallest operator count were re-run with a 600-second budget. Of the 14 targeted instances, 8 completed. Six remain solve-dominated. For example, *bottleneck* prob13 grounds in 26.4 seconds and solves in 101.0 seconds, *cave-diving* prob06 grounds in 6.9 seconds and solves in 350.5 seconds, and *pegsol-row5* prob04 grounds in 8.5 seconds and solves in 212.5 seconds. One instance, *chessboard-pebbling* prob06, is the least solve-leaning of the probe instances, with 54.8 seconds of grounding and 104.0 seconds of solving. Grounding accounts for roughly 35% of total time, well above the median solve-dominated ratio. A second, *chessboard-pebbling* prob07, is grounding-bound, with 124.0 seconds of grounding alone exceeding the original 120-second limit before its 223.2-second solve phase could begin. Six instances still timed out at 600 seconds without producing a phase breakdown. These are *diagnosis* prob18 and prob20, *document-transfer* prob02 and prob04, *cave-diving* prob04, and *pegsol-row5* prob05. The full per-instance timings of these 14 probe runs are listed in Appendix B.

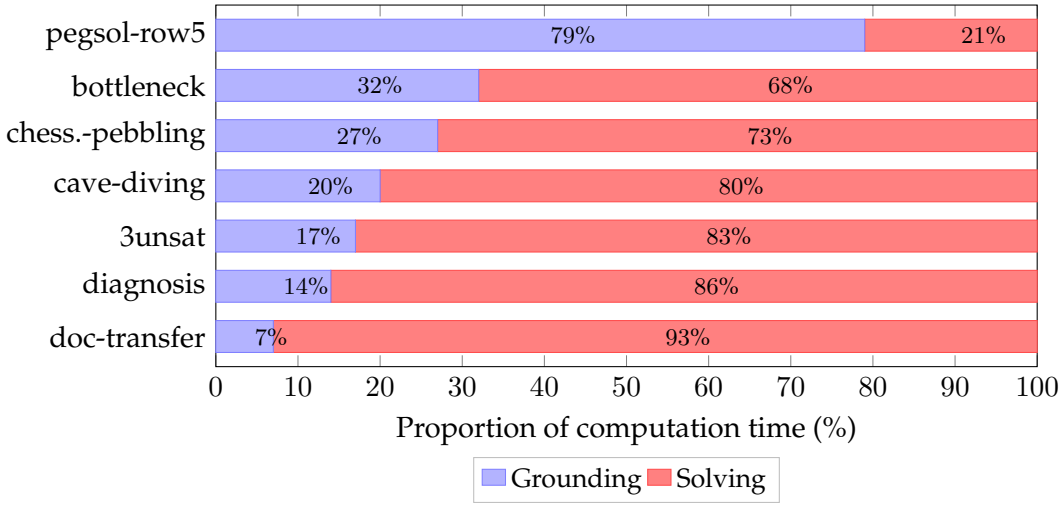

Figure 7: Median proportion of grounding and solving time per domain.

The 600-second probe runs support a refined picture of the bottleneck distribution. Solve-dominance extends well beyond the 120-second boundary into the 600-second budget for most of the tested domains, while grounding-boundedness is confined to the operator-heavy *chessboard-pebbling* and *pegsol-row5* instances, where the timestep-indexed encoding produces ground rules proportional to $|O| \times h$. The six unresolved 600-second timeouts remain unexplained by phase telemetry, so the two-pattern claim is empirically supported on the tested subset but not verified across all 90 timeouts.

Instance-Specific Solve Cost. Within each *3unsat* problem group, every instance shares the same encoding size, so grounding cost is effectively fixed and any variation in total runtime must come from the solving phase. This lets the sat-3-86-20 group act as a controlled comparison of instance-specific solve cost at fixed grounding cost. Figure 8 shows the five instances in the sat-3-86-20 group, which share identical problem structure ($|G| = 86$, $|P| = 106$, $|O| = 40$). Grounding time is effectively constant at 1.48 ± 0.02 seconds, confirming that grounding cost is determined by problem structure. Solving time, however, varies from 11.4 to 39.2 seconds (a factor of 3.5), reflecting the instance-specific combinatorial difficulty of the brave reasoning search across different clause assignments.

6.6. Discussion

This section interprets the empirical findings against the theoretical properties of Section 4 and Research Question 3 stated in Section 1. The discussion treats empirical verification of the proven measure relationships and bounds, the diagnostic utility provided by the scope-structure decomposition across categories of conflict,

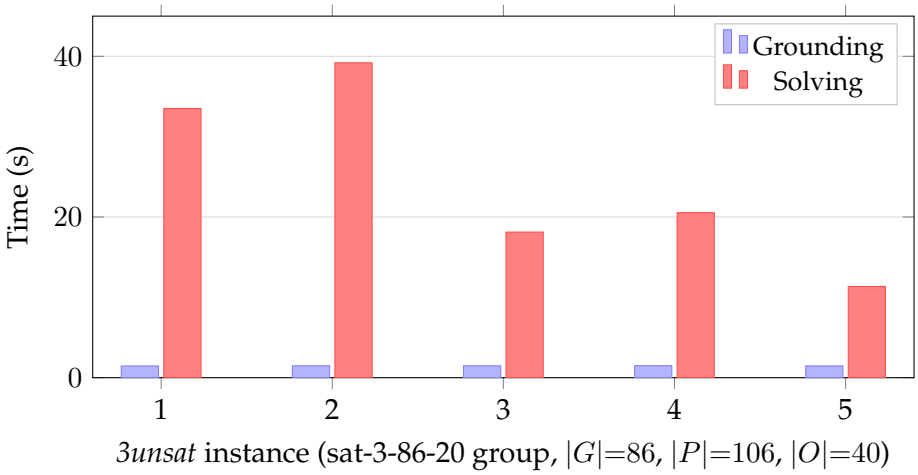

Figure 8: Grounding and solving time for five *3unsat* instances with identical problem structure.

and the limitations that bound the scope of any conclusion drawn from the present benchmark coverage.

6.6.1. Empirical Verification of Theoretical Properties

The benchmark results empirically verify all measure relationships from Proposition 15 and all scope-structure bounds from Proposition 16. Across all 68 computed instances, unreachable goals never participate in mutex or sequencing conflicts, and every instance with $\mathcal{I}_{\text{GS}}^{\text{scope}}(\Pi, 20) > 0$ also has $\mathcal{I}_{\text{MX}}^{\text{scope}}(\Pi, 20) > 0$. The upper bound $\mathcal{I}_{\text{MX}}^{\text{struct}}(\Pi, 20) \leq \binom{\mathcal{I}_{\text{MX}}^{\text{scope}}(\Pi,20)}{2}$ is achieved tightly for *diagnosis* prob16, where $\mathcal{I}_{\text{MX}}^{\text{struct}}(\text{prob16}, 20) = 28 = \binom{8}{2}$.

6.6.2. Diagnostic Utility

The measures address Research Question 3c by providing three levels of diagnostic information for unsolvable planning problems. At the coarsest level, category classification distinguishes unreachability from mutex and sequencing conflicts. This distinction is visible across domains, as *diagnosis* exhibits primarily unreachability while *bottleneck* and *chessboard-pebbling* exhibit primarily sequencing. At a finer level, scope measures quantify how many goals are affected by each conflict type. For instance, *diagnosis* prob06 has only 1 unreachable goal out of 7, representing 14% of goals, while prob05 has 49 out of 56, representing 88%, indicating fundamentally different repair difficulties. At the finest level, structure measures quantify the density of conflicts among affected goals. The contrast between *diagnosis* prob10 with $\mathcal{I}_{\text{MX}}^{\text{struct}}(\text{prob10}, 20) = 1$ and prob16 with $\mathcal{I}_{\text{MX}}^{\text{struct}}(\text{prob16}, 20) = 28$ illustrates how structure measures distinguish isolated conflicts from complete conflict graphs.

6.6.3. Limitations

Several limitations must be noted.

Domain Coverage. The measures are computable for 6 of 15 IPC 2016 domains and 1 additional Eriksson domain, with 68 of 158 unsolvable instances completing within the time limit. As shown in Section 6.5.2, the scalability limitation has two distinct causes: for completing instances, the solving phase dominates and its cost depends on instance-specific combinatorial difficulty; for instances exceeding the time limit, the grounding phase likely dominates due to the $O(|O| \times h)$ scaling of the timestep-indexed encoding. Lifted ASP solving techniques could potentially address the grounding bottleneck, while incremental solving strategies could reduce solve-phase cost, but both approaches are beyond the scope of this thesis.

Solvable Instance Validation. Only two solvable instances complete successfully, *cave-diving* `satprob01` and *document-transfer* `satprob01`, both correctly classified as consistent at $h = 20$. The remaining solvable instances fail for two distinct reasons. The *diagnosis* solvable instances and several *cave-diving* solvable instances fail at the *plasp* translation level due to predicate mismatches. The pipeline detects these failures through error handling and reports them as incompatible, but cannot resolve them automatically since the mismatch originates in the external *plasp* translator. Solvable instances in other domains exceed the 120-second time limit. A larger sample of solvable instances would strengthen the validation of the false-positive rate. Observing zero misclassifications across only two solvable instances provides weak statistical evidence. Even a procedure that misclassifies the majority of solvable problems could produce this outcome by chance, so the absence of false positives here cannot rule out higher error rates on larger samples.

Horizon Dependence. As demonstrated in Section 6.4, results depend on the horizon parameter. No efficiently computable sufficient horizon exists, and the appropriate value depends on the maximum meaningful plan length for a given domain. The default of $h = 20$ is empirically justified by the absence of false positives, but different domains may benefit from different horizon values.

Consistent Classifications. Thirty unsolvable instances are classified as consistent at $h = 20$. This does not indicate a measurement error. In both cases, the instances have no pairwise structural conflicts, and the consistent classification is the correct result of pairwise analysis. The unsolvability of these instances arises from properties that pairwise measures cannot detect. For the 10 IPC 2016 instances from *cave-diving* and *bottleneck*, the consistent classification at $h = 20$ is stable in the sense that shorter horizons produce spurious conflict detections indistinguishable from false positives on solvable instances (Section 6.4), confirming that no actual pairwise conflicts exist. Their unsolvability likely stems from global constraints, such

as resource capacities in *cave-diving* and spatial blocking in *bottleneck*, that require analysis beyond pairs. For the 20 *3unsat* instances, the consistent classification is horizon-independent: every pair of goal clauses is satisfiable in a common state, but no single variable assignment satisfies all clauses (Section 6.3.6).

Pairwise Limitation. The *3unsat* domain (Section 6.3.6) provides the first empirical evidence that pairwise goal analysis cannot detect unsolvability arising from the interaction of three or more goals. Extending the measures to k-tuple analysis would address this gap but at significantly higher computational cost.

Having established the practical diagnostic value of the measures and the limitations that bound it, Section 7 consolidates these findings, revisits the research questions, and outlines directions for future work.

7. Conclusion

This thesis investigated the adaptation of inconsistency measurement from propositional logic to classical planning problems, motivated by the observation that existing planning systems provide little structured diagnostic information when encountering unsolvable instances.

7.1. Summary of Contributions

The central research question asked how inconsistency measures can be meaningfully adapted to provide diagnostic value for unsolvable planning problems. The following subsections address the three research questions and summarize the contributions and limitations of this work.

7.1.1. Theoretical Adaptation (Research Question 1)

The theoretical investigation (Section 4) began with the observation that naive propositional encodings of planning problems yield uninformative measure values. The contension measure $\mathcal{I}_c$ returns 1 regardless of problem structure, failing to distinguish between a single missing operator, a pair of mutually exclusive goals, and an irreversible sequencing conflict (Section 4.2). This finding motivated the development of planning-native measures that operate directly on planning structures rather than reducing to propositional logic.

Three planning-native diagnostic profiles comprising six measures were proposed, organized by the structural phenomena they detect. The Unreachability Profile $\mathcal{I}_{\mathrm{UR}}$ characterizes dependency structure failures by counting goal propositions absent from all reachable states and the total number of unreachable propositions, revealing cascading failure depth. The Goal Mutex Profile $\mathcal{I}_{\mathrm{MX}}$ identifies operator-induced mutual exclusions where goals are individually achievable but never satisfiable in a

common reachable state, counting both the number of affected goals and the number of conflicting pairs. The Goal Sequencing Conflict Profile $\mathcal{I}_{\text{GS}}$ detects irreversible exclusions where achieving certain goals permanently blocks others, using ordered pairs to capture directional blocking relationships.

The measures were analyzed against rationality postulates adapted for the planning domain (Section 4.5.1). Classical monotonicity was reversed for Operator Monotonicity, reflecting that adding operators can only reduce inconsistency. All six measure components satisfy Goal Monotonicity, the requirement that adding a goal cannot decrease inconsistency, but none satisfies Planning Consistency, the requirement that the measure is zero exactly for solvable problems at sufficiently large horizons, confirming the necessity of the multi-faceted diagnostic approach. Formal relationships between the measures were also established. Sequencing conflicts imply mutex, unreachable goals are excluded from both conflict analyses, and the scope-structure bounds constrain the relationship between the two components of each profile (Section 4.5.2).

Computational complexity analysis revealed a uniform PSPACE-complete landscape. The decision problems UPPER, LOWER, and EXACT for all six measures are PSPACE-complete, and the function problems VALUE are in FPSPACE (Section 4.5.3). This uniformity contrasts with the multi-tier structures observed for propositional measures by Thimm and Wallner [39] and for temporal logic measures by Corea et al. [7]. The flat landscape reflects the fact that PLANSAT is already PSPACE-complete, subsuming the polynomial differences between measure definitions.

7.1.2. Computational Implementation (Research Question 2)

The measures were implemented (Section 5) using ASP, whose support for nonmonotonic reasoning fits the planning context where adding operators can reduce inconsistency (Section 4.5.1). The choice also builds on findings by Kuhlmann et al. [29] that ASP-based approaches show promise for inconsistency computation. The implementation follows a three-phase pipeline. PDDL files are first translated into ASP facts via *plasp* 3 by Dimopoulos et al. [9], then subjected to horizon-bounded state-space exploration via Clingo brave reasoning, and finally processed by a Python module that extracts measure values through set operations on the collected atoms (Section 5.1).

The key implementation insight is that brave reasoning computes the union of atoms true in any answer set without enumerating individual answer sets. Different answer sets correspond to different execution traces, and three categories of witness atoms capture the reachability and co-occurrence properties required by the measure definitions. The absence of a witness atom under brave reasoning constitutes a universal proof that the corresponding condition is unsatisfiable across all execution traces, providing the detection mechanism for mutex and sequencing conflicts (Section 5.3.3).

Correctness was verified through 80 automated tests covering scenario-level validation, hierarchy relationship tests, library-API contract tests, and end-to-end PDDL pipeline tests. All eleven test scenarios produced exact matches with their theoretically derived expected profiles (Section 5.6).

7.1.3. Practical Evaluation (Research Question 3)

Experimental evaluation (Section 6) on the IPC 2016 Unsolvability Competition benchmarks by Muise and Lipovetzky [32] and the unsolvable benchmark collection by Eriksson [13] demonstrated that the measures provide practical diagnostic value across seven compatible domains, namely six IPC domains and one Eriksson domain. Of 158 unsolvable instances, 68 (43%) completed within the 120-second time limit at horizon $h = 20$, yielding diagnostic profiles that distinguish between structurally different categories of unsolvable planning problems (Section 6.2).

The measures provide diagnostic information at three levels of granularity. Category classification distinguishes unreachability from mutex and sequencing conflicts. The *diagnosis* domain exhibits primarily unreachability, while *bottleneck* and *chessboard-pebbling* exhibit primarily sequencing. Scope measures quantify severity, ranging from a single unreachable goal in *diagnosis* prob06, where 14% of goals are affected, to near-total unreachability in prob05, where 88% are affected. Structure measures reveal conflict density, distinguishing isolated pairwise conflicts in *diagnosis* prob10 from complete conflict graphs in prob16, where every pair of affected goals is in conflict.

The horizon sensitivity analysis (Section 6.4) revealed a soundness-coverage tradeoff inherent to horizon-bounded brave reasoning. Shorter horizons risk false positives, as demonstrated by the solvable *cave-diving* `satprob01` being spuriously classified as 2c-sequencing, the pairwise sequencing category, at $h = 10$. Longer horizons increase grounding and solving cost, reducing the number of instances that complete within the time limit. The default of $h = 20$ avoids all observed false positives while detecting non-zero conflict profiles in 38 of 68 unsolvable instances.

The *3unsat* domain from the Eriksson benchmarks [13] provided the first empirical evidence for a fundamental limitation of pairwise analysis. All 20 completed instances return zero profiles at every horizon because their unsolvability arises from the interaction of three or more goals (Section 6.3.6).

7.2. Limitations

The principal limitation is scalability. The grounding bottleneck restricts applicability to problems of moderate size, with 57% of benchmark instances exceeding the 120-second time limit at $h = 20$. Only 6 of 15 IPC 2016 domains are compatible with the pipeline, with two excluded due to unsupported PDDL features and seven due to grounding complexity. The dependence on the horizon parameter h introduces a further practical concern. No efficiently computable sufficient horizon exists, and different domains may require different values.

The pairwise nature of the mutex and sequencing measures constitutes a theoretical limitation. For problems where every goal pair is achievable in some state but the full goal set is not, the measures return zero profiles regardless of horizon. The *3unsat* benchmark results confirm that such problems exist in practice.

Finally, only two solvable instances completed successfully, limiting the empirical validation of the false-positive rate. A larger sample of solvable instances would strengthen confidence in the measures' specificity.

7.3. Closing Assessment

This thesis demonstrates that inconsistency measurement can be meaningfully adapted from propositional logic to classical planning when the adaptation respects planning-specific semantics. The key methodological insight is that planning conflicts manifest as reachability and achievability failures rather than as direct logical contradictions, and that meaningful diagnostics require operating directly on planning structures rather than reducing to propositional encodings. The resulting measures provide a quantitative vocabulary for describing unsolvable planning problems, capturing how many goals are affected, what type of conflict prevents their satisfaction, and how densely the conflicts are interconnected. This vocabulary addresses a methodological gap in automated planning tools, where unsolvable instances have traditionally been met with binary failure reports rather than structured diagnostic information.

7.4. Future Work

Several directions merit further investigation.

Higher-Order Goal Analysis. The proposed measures detect pairwise conflicts, specifically mutex relationships between goal pairs and sequencing conflicts between ordered goal pairs. For problems with two or fewer goals ($|G| \leq 2$), the conjunction of all measures equaling zero implies solvability. However, for problems with three or more goals, pairwise achievability does not guarantee that the full goal set is achievable. Consider goals $G = \{g_1, g_2, g_3\}$ with reachable states $\{g_1, g_2\}$, $\{g_2, g_3\}$, and $\{g_1, g_3\}$. Every pair coexists in some state, yet no state satisfies all three goals. Extending the mutex definition to k-tuples would address this limitation, analogous to the distinction between 2-SAT and k-SAT in satisfiability theory.

Extension Beyond STRIPS. The proposed measures operate on STRIPS-style classical planning (Section 2.2). Numeric planning, temporal planning, and conditional effects are common PDDL extensions for which inconsistency measurement remains open. Adapting the measures to handle these features, possibly drawing on

the temporal logic adaptation by Corea et al. [7] for time-sensitive constructs, would broaden applicability beyond classical planning.

Restricted Planning Fragments. Bylander [6] identifies planning restrictions for which PLANSAT drops from PSPACE-complete to NP-complete or polynomial-time. Analyzing how such restrictions affect the complexity of the proposed inconsistency measures could produce a multi-tier structure where different measures exhibit different complexity, as observed for propositional measures by Thimm and Wallner [39].

Alternative Encodings for Scalability. The grounding bottleneck causes 57% of benchmark instances to exceed the time limit at $h = 20$ (Section 6.5). Niskanen et al. [33] demonstrate that MaxSAT-based approaches can outperform ASP for some propositional inconsistency measures. Adapting MaxSAT or symbolic state-space methods to the planning-native measures, or exploring lifted reasoning to avoid full grounding, could extend applicability to larger instances.

Adaptive Horizon Selection. Section 6.4 establishes a default h and heuristic mitigation strategies for the soundness-coverage tradeoff. A formal iterative-deepening procedure that increases h until unreachability stabilizes, exploiting Proposition 2, or domain-specific heuristics for sufficient horizon estimation, would automate this choice.

Diagnostic-Driven Repair. The proposed measures explain why a planning problem is unsolvable but do not suggest how to make it solvable. Unreachable goal propositions identify missing dependency chains, mutex pairs indicate operator conflicts to relax, and sequencing conflicts mark irreversible transitions to redesign. Coupling these diagnostics with minimal-modification repair, in the style of Göbelbecker et al. [23], would extend the work from diagnosis to repair.

References

[1] Mohammad Abdulaziz, Michael Norrish, and Charles Gretton. Formally verified algorithms for upper-bounding state space diameters. *Journal of Automated Reasoning*, 61(1-4):485–520, February 2018.

[2] Philippe Besnard and John Grant. Relative inconsistency measures. *Artificial Intelligence*, 280:103231, March 2020.

[3] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, February 1997.

[4] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, June 2001.

[5] Sebastian Bunge. Planning inconsistency measures. `https://github.com/BlueberryDuck/planning-inconsistency-measures`, April 2026. Version 1.1.0, archived at `https://doi.org/10.5281/zenodo.19908045`.

[6] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2):165–204, September 1994.

[7] Carl Corea, John Grant, and Matthias Thimm. Measuring inconsistency in declarative process specifications. In Claudio Di Ciccio, Remco M. Dijkman, Adela del Río-Ortega, and Stefanie Rinderle-Ma, editors, *Business Process Management - 20th International Conference, BPM 2022, Proceedings*, volume 13420 of *Lecture Notes in Computer Science*, pages 289–306, Münster, Germany, September 2022. Springer.

[8] Carl Corea, Isabelle Kuhlmann, Matthias Thimm, and John Grant. Paraconsistent reasoning for inconsistency measurement in declarative process specifications. *Information Systems*, 122:102347, May 2024.

[9] Yannis Dimopoulos, Martin Gebser, Patrick Lühne, Javier Romero, and Torsten Schaub. plasp 3: Towards effective ASP planning. *Theory and Practice of Logic Programming*, 19(3):477–504, May 2019.

[10] Yannis Dimopoulos, Bernhard Nebel, and Jana Koehler. Encoding planning problems in nonmonotonic logic programs. In Sam Steel and Rachid Alami, editors, *Recent Advances in AI Planning, 4th European Conference on Planning, ECP'97, Proceedings*, volume 1348 of *Lecture Notes in Computer Science*, pages 169–181, Toulouse, France, September 1997. Springer.

[11] Stefan Edelkamp and Jörg Hoffmann. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, Institut für Informatik, Albert-Ludwigs-Universität Freiburg, January 2004.

[12] Thomas Eiter and Georg Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, 15(3-4):289–323, September 1995.

[13] Salomé Eriksson. Unsolvable PDDL benchmarks. Zenodo dataset, `https://doi.org/10.5281/zenodo.3355446`, July 2019. Version 1, accompanying the PhD thesis of Salomé Eriksson, University of Basel.

[14] Salomé Eriksson, Gabriele Röger, and Malte Helmert. A proof system for unsolvable planning tasks. In Mathijs de Weerdt, Sven Koenig, Gabriele Röger, and Matthijs T. J. Spaan, editors, *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018*, pages 65–73, Delft, The Netherlands, June 2018. Association for the Advancement of Artificial Intelligence (AAAI).

[15] Marco Favorito, Francesco Fuggitti, and Christian Muise. pddl: An unquestionable PDDL 3.1 parser. `https://github.com/AI-Planning/pddl`, 2021. Python package, version 0.4.5.

[16] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208, December 1971.

[17] Maria Fox and Derek Long. PDDL+: Modelling continuous time-dependent effects. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, volume 4, pages 34–43, 2002.

[18] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, December 2003.

[19] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187-188:52–89, August 2012.

[20] Alfonso Gerevini and Derek Long. Plan constraints and preferences in PDDL3: The language of the fifth international planning competition. Technical report, Department of Electronics for Automation, University of Brescia, Brescia, Italy, August 2005.

[21] Malik Ghallab, Adele Howe, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL - the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, October 1998.

[22] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning: Theory and practice*. The Morgan Kaufmann Series in Artificial Intelligence. Elsevier, 2004.

[23] Moritz Göbelbecker, Thomas Keller, Patrick Eyerich, Michael Brenner, and Bernhard Nebel. Coming up with good excuses: What to do when no plan can be found. In Ronen I. Brafman, Hector Geffner, Jörg Hoffmann, and Henry A. Kautz, editors, *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS*, volume 20, pages 81–88, Toronto, Ontario, Canada, May 2010. Association for the Advancement of Artificial Intelligence (AAAI).

[24] John Grant and Anthony Hunter. Measuring consistency gain and information loss in stepwise inconsistency resolution. In Weiru Liu, editor, *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 11th European Conference, ECSQARU 2011, Proceedings*, volume 6717 of *Lecture Notes in Computer Science*, pages 362–373, Belfast, UK, June 2011. Springer.

[25] John Grant and Anthony Hunter. Semantic inconsistency measures using 3-valued logics. *International Journal of Approximate Reasoning*, 156:38–60, May 2023.

[26] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, May 2001.

[27] Anthony Hunter and Sébastien Konieczny. Measuring inconsistency through minimal inconsistent sets. In Gerhard Brewka and Jérôme Lang, editors, *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning*, pages 358–366, Sydney, NSW, Australia, September 2008. Association for the Advancement of Artificial Intelligence (AAAI).

[28] Isabelle Kuhlmann. A MaxSAT-based approach for computing inconsistency degrees in linear temporal logic on fixed traces. In Carl Corea, Jérôme Delobelle, John Grant, Jean-Guy Mailly, Julien Rossit, and Kenneth Skiba, editors, *Joint Proceedings of the ECSQARU 2025 Workshops and Tutorials*, pages 59–69, Hagen, Germany, September 2025. HAL. hal-05294280.

[29] Isabelle Kuhlmann, Anna Gessler, Vivien Laszlo, and Matthias Thimm. Comparison of SAT-based and ASP-based algorithms for inconsistency measurement. *Journal of Artificial Intelligence Research*, 82:563–685, February 2025.

[30] Isabelle Kuhlmann, Mark O. Mints, Peer Neubert, and Matthias Thimm. Diagnosing non-executable plans via inconsistency metrics: A KR-inspired perspective for cognitive robotics. In *The 13th International Cognitive Robotics Workshop (CogRob'25)*, November 2025.

[31] Vladimir Lifschitz. *Answer Set Programming*. Springer, Cham, 2019.

[32] Christian Muise and Nir Lipovetzky. Unsolvability international planning competition. `https://unsolve-ipc.eng.unimelb.edu.au/`, 2016. Competition held at ICAPS 2016, London, UK.

[33] Andreas Niskanen, Isabelle Kuhlmann, Matthias Thimm, and Matti Järvisalo. MaxSAT-based inconsistency measurement. In Kobi Gal, Ann Nowé, Grzegorz J. Nalepa, Roy Fairstein, and Roxana Radulescu, editors, *ECAI 2023 – 26th European Conference on Artificial Intelligence, Including 12th Conference on Prestigious Applications of Intelligent Systems (PAIS 2023)*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, pages 1779–1786, Kraków, Poland, September 2023. IOS Press.

[34] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, MA, 1994. Reprinted with corrections, 1995.

[35] Graham Priest. The logic of paradox. *Journal of Philosophical Logic*, 8:219–241, 1979.

[36] Stuart J. Russell and Peter Norvig. *Artificial intelligence*. Prentice Hall Series in Artificial Intelligence. Pearson, Harlow, third edition, global edition edition, 2016.

[37] Sarath Sreedharan, Siddharth Srivastava, David E. Smith, and Subbarao Kambhampati. Why can't you do that HAL? explaining unsolvability of planning tasks. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019*, pages 1422–1430, Macao, China, August 2019. ijcai.org.

[38] Matthias Thimm. Inconsistency measurement. In Nahla Ben Amor, Benjamin Quost, and Martin Theobald, editors, *Scalable Uncertainty Management - 13th International Conference, SUM 2019, Proceedings*, volume 11940 of *Lecture Notes in Computer Science*, pages 9–23, Compiègne, France, December 2019. Springer.

[39] Matthias Thimm and Johannes Peter Wallner. Some complexity results on inconsistency measurement. In Chitta Baral, James P. Delgrande, and Frank Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016*, pages 114–124, Cape Town, South Africa, April 2016. Association for the Advancement of Artificial Intelligence (AAAI).

[40] Markus Ulbricht, Matthias Thimm, and Gerhard Brewka. Measuring inconsistency in answer set programs. In Loizos Michael and Antonis C. Kakas, editors, *Logics in Artificial Intelligence - 15th European Conference, JELIA 2016, Proceedings*, volume 10021 of *Lecture Notes in Computer Science*, pages 577–583, Larnaca, Cyprus, November 2016. Springer.

A. Full Benchmark Results

This appendix lists the per-instance results of the primary benchmark run summarized in Section 6.2. The CSV outputs used here are released with the implementation by Sebastian Bunge [5]. The primary run was executed on 2026-04-02 with horizon $h = 20$ and a 120-second time limit per instance. Each table reports the problem identifier, problem size ($|G|$, $|P|$, $|O|$), scope and structural values for the three profiles, the per-phase times (translation, grounding, solving, extraction), and the total runtime. Solvable validation instances (`satprob*`), instances of unknown solvability (`unknownprob*`), and instances rejected by *plasp* translation are omitted, as documented in the data release.

A.1. 3unsat

	Size			UR		MX		GS		Time (s)				
Problem	$\|G\|$	$\|P\|$	$\|O\|$	$\mathcal{I}^s$	$\mathcal{I}^t$	$\mathcal{I}^s$	$\mathcal{I}^t$	$\mathcal{I}^s$	$\mathcal{I}^t$	**Tr**	**Gr**	**So**	**Ex**	**Tot**
sat-3-22-5-1	22	27	10	0	0	0	0	0	0	0.005	0.059	0.220	0.000	0.290
sat-3-22-5-2	22	27	10	0	0	0	0	0	0	0.005	0.058	0.207	0.000	0.276
sat-3-22-5-3	22	27	10	0	0	0	0	0	0	0.005	0.062	0.247	0.000	0.319
sat-3-22-5-4	22	27	10	0	0	0	0	0	0	0.004	0.057	0.205	0.000	0.272
sat-3-22-5-5	22	27	10	0	0	0	0	0	0	0.005	0.060	0.230	0.000	0.300
sat-3-43-10-1	43	53	20	0	0	0	0	0	0	0.006	0.280	1.183	0.001	1.486
sat-3-43-10-2	43	53	20	0	0	0	0	0	0	0.006	0.281	1.322	0.001	1.626
sat-3-43-10-3	43	53	20	0	0	0	0	0	0	0.007	0.281	1.249	0.001	1.552
sat-3-43-10-4	43	53	20	0	0	0	0	0	0	0.007	0.266	1.316	0.001	1.605
sat-3-43-10-5	43	53	20	0	0	0	0	0	0	0.006	0.274	1.122	0.001	1.417
sat-3-65-15-1	65	80	30	0	0	0	0	0	0	0.008	0.749	3.612	0.002	4.402
sat-3-65-15-2	65	80	30	0	0	0	0	0	0	0.009	0.736	3.312	0.002	4.086
sat-3-65-15-3	65	80	30	0	0	0	0	0	0	0.008	0.754	4.181	0.002	4.976
sat-3-65-15-4	65	80	30	0	0	0	0	0	0	0.009	0.756	9.634	0.002	10.436
sat-3-65-15-5	65	80	30	0	0	0	0	0	0	0.009	0.769	4.532	0.002	5.342
sat-3-86-20-1	86	106	40	0	0	0	0	0	0	0.010	1.463	33.514	0.004	35.056
sat-3-86-20-2	86	106	40	0	0	0	0	0	0	0.010	1.480	39.195	0.004	40.765
sat-3-86-20-3	86	106	40	0	0	0	0	0	0	0.011	1.481	18.121	0.003	19.671
sat-3-86-20-4	86	106	40	0	0	0	0	0	0	0.010	1.496	20.541	0.003	22.112
sat-3-86-20-5	86	106	40	0	0	0	0	0	0	0.011	1.467	11.359	0.004	12.894
sat-3-108-25-1								TIMEOUT						
sat-3-108-25-2								TIMEOUT						
sat-3-108-25-3								TIMEOUT						
sat-3-108-25-4								TIMEOUT						
sat-3-108-25-5								TIMEOUT						
sat-3-129-30-1								TIMEOUT						
sat-3-129-30-2								TIMEOUT						
sat-3-129-30-3								TIMEOUT						
sat-3-129-30-4								TIMEOUT						
sat-3-129-30-5								TIMEOUT						

Table 16: Per-instance results for *3unsat*.

A.2. Bottleneck

	Size			UR		MX		GS		Time (s)				
Problem	$\|G\|$	$\|P\|$	$\|O\|$	$\mathcal{I}^s$	$\mathcal{I}^t$	$\mathcal{I}^s$	$\mathcal{I}^t$	$\mathcal{I}^s$	$\mathcal{I}^t$	**Tr**	**Gr**	**So**	**Ex**	**Tot**
prob01	4	629	4913	0	533	4	6	4	12	0.005	0.825	1.257	0.000	2.091
prob02	4	702	5832	0	594	3	3	3	6	0.005	0.950	1.374	0.000	2.333
prob03	4	779	6859	0	659	2	1	2	2	0.005	1.151	1.637	0.000	2.797
prob04	5	1430	17576	0	1247	5	10	5	20	0.005	3.337	7.054	0.000	10.402
prob05	5	1539	19683	0	1343	4	6	4	12	0.006	3.938	7.415	0.000	11.368
prob06	5	1652	21952	0	1443	3	3	3	6	0.005	3.957	7.590	0.000	11.561
prob07	5	1769	24389	0	1547	2	1	2	2	0.007	4.600	9.910	0.000	14.528
prob08	6	2849	50653	0	2541	6	15	6	30	0.006	9.692	22.386	0.000	32.098
prob09	6	3002	54872	0	2680	4	4	4	8	0.006	10.616	36.020	0.000	46.660
prob10	6	3159	59319	0	2823	2	1	2	2	0.006	12.731	39.098	0.000	51.853
prob11	6	3320	64000	0	2970	0	0	0	0	0.007	13.800	48.306	0.000	62.130
prob12	6	3485	68921	0	3121	0	0	0	0	0.007	13.987	53.781	0.000	67.794
prob13								TIMEOUT						
prob14								TIMEOUT						
prob15								TIMEOUT						
prob16								TIMEOUT						
prob17								TIMEOUT						
prob18								TIMEOUT						
prob19								TIMEOUT						
prob20								TIMEOUT						
prob21								TIMEOUT						
prob22								TIMEOUT						
prob23								TIMEOUT						
prob24								TIMEOUT						
prob25								TIMEOUT						

Table 17: Per-instance results for *bottleneck*.

A.3. Cave-Diving

	Size			UR		MX		GS		Time (s)				
Problem	$\|G\|$	$\|P\|$	$\|O\|$	$\mathcal{I}^s$	$\mathcal{I}^t$	$\mathcal{I}^s$	$\mathcal{I}^t$	$\mathcal{I}^s$	$\mathcal{I}^t$	**Tr**	**Gr**	**So**	**Ex**	**Tot**
prob01	3	161	4732	0	78	0	0	0	0	0.005	0.774	1.745	0.000	2.534
prob02	3	141	3758	0	66	0	0	0	0	0.005	0.564	1.055	0.000	1.631
prob03	3	123	2884	0	56	0	0	0	0	0.005	0.421	0.843	0.000	1.276
prob04								TIMEOUT						
prob05	3	81	862	1	49	0	0	0	0	0.005	0.102	0.433	0.000	0.545
prob06								TIMEOUT						
prob07	6	389	27884	0	186	0	0	0	0	0.006	4.954	34.965	0.000	39.956
prob08	6	269	16056	0	117	0	0	0	0	0.006	2.993	13.323	0.001	16.344
prob09								TIMEOUT						
prob10								TIMEOUT						
prob11								TIMEOUT						
prob12	5	362	24996	0	177	0	0	0	0	0.006	4.392	27.933	0.001	32.358
prob13	5	269	16056	0	117	0	0	0	0	0.006	3.054	14.520	0.000	17.600
prob14	5	194	8916	0	75	0	0	0	0	0.006	1.615	6.517	0.000	8.152
prob15								TIMEOUT						
prob16								TIMEOUT						
prob17								TIMEOUT						
prob18								TIMEOUT						
prob19								TIMEOUT						
prob20								TIMEOUT						
prob21								TIMEOUT						
prob22								TIMEOUT						
prob23								TIMEOUT						
prob24								TIMEOUT						
prob25								TIMEOUT						

Table 18: Per-instance results for *cave-diving*.

A.4. Chessboard-Pebbling

	Size			UR		MX		GS		Time (s)				
Problem	$\|G\|$	$\|P\|$	$\|O\|$	$\mathcal{I}^s$	$\mathcal{I}^t$	$\mathcal{I}^s$	$\mathcal{I}^t$	$\mathcal{I}^s$	$\mathcal{I}^t$	**Tr**	**Gr**	**So**	**Ex**	**Tot**
prob03	3	1300	15625	0	1211	3	2	3	2	0.004	3.018	7.340	0.000	10.368
prob04	3	2664	46656	0	2533	3	2	3	2	0.005	8.685	35.789	0.001	44.490
prob05	3	4900	117649	0	4719	3	2	3	2	0.007	22.790	61.681	0.001	84.496
prob06							TIMEOUT							
prob07							TIMEOUT							
prob08							TIMEOUT							
prob09							TIMEOUT							
prob10							TIMEOUT							
prob11							TIMEOUT							
prob12							TIMEOUT							
prob13							TIMEOUT							
prob14							TIMEOUT							
prob15							TIMEOUT							
prob16							TIMEOUT							
prob17							TIMEOUT							
prob18							TIMEOUT							
prob19							TIMEOUT							
prob20							TIMEOUT							
prob21							TIMEOUT							
prob22							TIMEOUT							
prob23							TIMEOUT							
prob24							TIMEOUT							
prob25							TIMEOUT							

Table 19: Per-instance results for *chessboard-pebbling*.

A.5. Diagnosis

	Size			UR		MX		GS		Time (s)				
Problem	$\|G\|$	$\|P\|$	$\|O\|$	$\mathcal{I}^s$	$\mathcal{I}^t$	$\mathcal{I}^s$	$\mathcal{I}^t$	$\mathcal{I}^s$	$\mathcal{I}^t$	**Tr**	**Gr**	**So**	**Ex**	**Tot**
prob01	16	190	216	9	16	0	0	0	0	0.035	0.062	0.364	0.000	0.493
prob02	16	190	216	9	16	0	0	0	0	0.035	0.065	0.361	0.000	0.494
prob03	16	155	125	10	20	0	0	0	0	0.020	0.034	0.147	0.000	0.219
prob04	38	389	568	31	59	0	0	0	0	0.104	0.262	1.085	0.000	1.553
prob05	56	542	957	49	97	0	0	0	0	0.151	0.347	2.295	0.001	2.941
prob06	7	49	25	1	1	0	0	0	0	0.006	0.010	0.018	0.000	0.038
prob07	22	154	81	12	31	0	0	0	0	0.013	0.024	0.102	0.000	0.149
prob08	24	170	96	14	41	0	0	0	0	0.019	0.036	0.128	0.000	0.198
prob09	24	241	156	14	33	0	0	0	0	0.034	0.113	42.967	0.000	43.151
prob10	24	187	99	14	25	2	1	2	2	0.020	0.040	0.187	0.000	0.264
prob11	22	204	139	14	33	0	0	0	0	0.020	0.050	2.692	0.000	2.782
prob12	24	199	99	15	26	0	0	0	0	0.018	0.045	0.411	0.000	0.491
prob13	36	399	232	27	67	0	0	0	0	0.091	0.199	0.573	0.000	0.947
prob14	28	314	152	19	38	0	0	0	0	0.041	0.114	1.157	0.000	1.357
prob15	24	240	155	14	35	0	0	0	0	0.028	0.059	0.405	0.000	0.517
prob16	36	457	257	20	32	8	28	8	56	0.106	0.266	2.077	0.001	2.542
prob17	24	214	120	15	27	0	0	0	0	0.023	0.049	0.359	0.000	0.451
prob18								TIMEOUT						
prob19	24	214	120	15	27	0	0	0	0	0.023	0.046	0.376	0.000	0.465
prob20								TIMEOUT						

Table 20: Per-instance results for *diagnosis*.

A.6. Document-Transfer

	Size			UR		MX		GS		Time (s)				
Problem	$\|G\|$	$\|P\|$	$\|O\|$	$\mathcal{I}^s$	$\mathcal{I}^t$	$\mathcal{I}^s$	$\mathcal{I}^t$	$\mathcal{I}^s$	$\mathcal{I}^t$	**Tr**	**Gr**	**So**	**Ex**	**Tot**
prob01	19	1680	25839	19	1580	0	0	0	0	0.006	3.514	9.704	0.001	13.264
prob02								TIMEOUT						
prob03	19	1425	20592	19	1332	0	0	0	0	0.006	2.755	18.691	0.001	21.489
prob04								TIMEOUT						
prob05								TIMEOUT						
prob06	4	210	1188	1	95	0	0	0	0	0.004	0.122	18.800	0.000	18.935
prob07								TIMEOUT						
prob08								TIMEOUT						
prob09								TIMEOUT						
prob10								TIMEOUT						
prob11								TIMEOUT						
prob12								TIMEOUT						
prob13								TIMEOUT						
prob14								TIMEOUT						
prob15								TIMEOUT						
prob16								TIMEOUT						
prob17								TIMEOUT						
prob18								TIMEOUT						
prob19								TIMEOUT						
prob20								TIMEOUT						

Table 21: Per-instance results for *document-transfer*.

A.7. Pegsol-Row5

	Size			UR		MX		GS		Time (s)				
Problem	$\|G\|$	$\|P\|$	$\|O\|$	$\mathcal{I}^s$	$\mathcal{I}^t$	$\mathcal{I}^s$	$\mathcal{I}^t$	$\mathcal{I}^s$	$\mathcal{I}^t$	**Tr**	**Gr**	**So**	**Ex**	**Tot**
prob01	1	228	216	1	214	0	0	0	0	0.009	0.019	0.004	0.000	0.036
prob02	1	2772	2744	1	2732	0	0	0	0	0.004	0.367	0.097	0.000	0.470
prob03	1	13872	13824	1	13782	0	0	0	0	0.005	2.173	1.901	0.001	4.084
prob04								TIMEOUT						
prob05								TIMEOUT						
prob06								TIMEOUT						
prob07								TIMEOUT						
prob08								TIMEOUT						
prob09								TIMEOUT						
prob10								TIMEOUT						
prob11								TIMEOUT						
prob12								TIMEOUT						
prob13								TIMEOUT						
prob14								TIMEOUT						
prob15								TIMEOUT						

Table 22: Per-instance results for *pegsol-row5*.

B. Extended-Budget Runs

This appendix lists the per-instance results for the extended-budget probe runs discussed in Section 6.5. The two smallest timed-out instances per domain (14 instances in total) were re-executed with a 600-second time limit to recover measure values and a phase breakdown that the 120-second timeout leaves unreported. Six of the 14 runs again exhausted the 600-second limit and report no breakdown.

		Size			UR		MX		GS		Time (s)				
Domain	**Problem**	$\|G\|$	$\|P\|$	$\|O\|$	$\mathcal{I}^s$	$\mathcal{I}^t$	$\mathcal{I}^s$	$\mathcal{I}^t$	$\mathcal{I}^s$	$\mathcal{I}^t$	**Tr**	**Gr**	**So**	**Ex**	**Tot**
3unsat	sat-3-108-25-1	108	133	50	0	0	0	0	0	0	0.019	2.157	139.025	0.006	141.375
3unsat	sat-3-108-25-2	108	133	50	0	0	0	0	0	0	0.013	2.169	181.972	0.006	184.352
bottleneck	prob13	7	5150	125000	0	4673	7	21	7	42	0.006	26.381	101.037	0.001	127.448
bottleneck	prob14	7	5355	132651	0	4863	7	12	7	24	0.006	27.930	182.072	0.001	210.037
cave-diving	prob04								TIMEOUT						
cave-diving	prob06	6	473	35736	0	260	0	0	0	0	0.005	6.918	350.532	0.000	357.488
chessboard-pebbling	prob06	3	8320	262144	0	8081	3	2	3	2	0.007	54.755	103.956	0.001	158.748
chessboard-pebbling	prob07	3	13284	531441	0	12979	3	2	3	2	0.009	124.007	223.198	0.002	347.364
diagnosis	prob18								TIMEOUT						
diagnosis	prob20								TIMEOUT						
document-transfer	prob02								TIMEOUT						
document-transfer	prob04								TIMEOUT						
pegsol-row5	prob04	1	46728	46656	1	46572	0	0	0	0	0.007	8.510	212.546	0.003	221.081
pegsol-row5	prob05								TIMEOUT						

Table 23: Per-instance results for the extended-budget probe runs.

C. Reproducibility

The implementation, encodings, batch runner, and raw result data underlying this appendix and Section 6 are publicly archived [5]. The Docker image bundled with the repository pins the toolchain to Python 3.12, Clingo 5.8.0, and *plasp* 3.1.1. All measurements were taken on a host with an AMD Ryzen 5 5600X CPU and 32 GiB of RAM, with Clingo invoked single-threaded through its Python API.

The primary results in Tables 16–22 were produced on 2026-04-02 with horizon $h = 20$ and a 120-second per-instance time limit using the batch runner

```
./run.sh planning-measures batch <domain-dir> \
    -o <out.csv> \
    -H 20 \
    -t 120
```

where `<domain-dir>` ranges over the IPC 2016 directories listed as compatible in Table 9, plus the Eriksson *3unsat* directory. The seven resulting CSV files share the schema documented in the data release: problem identifier, problem size, scope and structural values for the three measures, category, per-phase timings, and status. The extended-budget probe runs in Table 23 were produced on 2026-04-20 and 2026-04-21 by invoking the `compute` subcommand per instance with `-H 20` and `-t 600` on the 14 selected problems.