University of Basel

Empirical Analysis of the Tseitin Transformation on Axioms in Fast Downward

Runtime and Memory Consumption after Plaisted-Greenbaum Encoding of FDR Axioms

Bachelor Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Research Group Artificial Intelligence
https://ai.dmi.unibas.ch

Examiner: Prof. Dr. Malte Helmert
Supervisor: Gabriele Röger

Jan Daniele Walliser
jan.walliser@unibas.ch
22-057-954

23.08.2025

Abstract

It has recently been discovered that applying a Tseitin transformation to axioms can improve the overall coverage of benchmark domains. We ask whether this effect can be observed on the level of finite-domain representation (FDR). Specifically, we implement a Plaisted–Greenbaum (PG) encoding for FDR axioms in Fast Downward within the search component, integrated post-translation via a DelegatingTask wrapper so that all other components remain unchanged. We also prove that the induced layering remains valid: each auxiliary variable is assigned the maximum layer of its body variables, preserving stratified, terminating, and deterministic axiom evaluation.
There is ongoing debate whether axioms should be supported natively or compiled away. Our results add nuance: across standard axiom-rich benchmarks under greedy best-first search (eager) with FF and additive heuristics, the PG Tseitin transformation on FDR axioms does not reproduce the previously reported gains. Coverage is essentially unchanged; runtimes are similar or slightly worse; and peak memory can increase due to a larger number of (now smaller) axioms. Initial heuristic values remain unaffected.
We conclude that the benefits observed in prior work likely arise from transformations at different representation levels (e.g., symbolic or SAT-oriented encodings) rather than from FDR-time axiom restructuring. In its PG form, Tseitin on FDR axioms is not a general performance win; its advantages appear situational and do not materialize inside the FDR search layer.

Table of Contents

1 Introduction

1.1 Motivation

Automated planning is one of the central areas in Artificial Intelligence (AI) with a wide range of applications, for instance in robotics [5], logistics and transportation [20]. In a more general form, a planning problem consists of a description of the initial state of the world, a set of possible actions, as well as a goal specification. The goal of a planner is to find a sequence of actions that transforms the initial state into a state that satisfies the goal condition, a goal state. Over the past decades, planners became more and more efficient in terms of performance. One of those planning systems is *Fast Downward* [6], which also includes the feature of *axiom* support. Axioms allow the definition of facts that are not changed through actions of the planner but are derived from other facts through logical inference. In the Planning Domain Definition Language (PDDL), such derived facts are expressed using *derived predicates.* They are important for modelling indirect dependencies, complex domain rules as well as relationships that would otherwise require an explosion in the number of explicit action definitions. In modern planning systems, the axioms are evaluated using a layered *fixed point iteration.* Although axioms are essential for concise domain modelling [18], their evaluation overhead can dominate the computational cost in some problems. This has recently been highlighted in work by Borgwardt et al. [2], who investigated planning with *ontologies.* They showed that applying a Tseitin-style transformation to large axiom bodies can drastically increase the number of solved instances in benchmark domains. This raises the question whether similar effects occur in the context of *Fast Downward* and its axiom evaluation within the search component:

- Is the observed benefit due to the transformation itself, or to its interaction with the way axioms are compiled and evaluated?
- How does the transformation influence search time and memory usage?

To investigate this, we apply a Tseitin-inspired transformation directly at the level of axioms in finite-domain representation (FDR) inside the search component of *Fast Downward.* The idea is to replace “wide” axioms with equivalent sets of smaller axioms, thereby reducing the number of literals checked per axiom, at the cost of introducing more (auxiliary) axioms.

1.2 Objectives

The aim of this thesis is to investigate whether applying a Tseitin transformation in this manner improves the coverage of benchmarks domains and observe its influence on the overall performance of *Fast Downward*. Specifically, the objectives are:

- **Benchmark coverage:** Determine if the impact of a Tseitin transformation on FDR leads to an improvement on coverage over benchmark domains such as shown by Borgwardt et al. [2].
- **Robustness across configurations:** Compare the results under two heuristic functions (*additive*, *FF*), on a greedy best-first search in its eager variant (*eager greedy*) and with two axiom evaluation modes within the heuristics (*approximate no negative*, *approximate no negative cycles*) to determine whether the impact discovered by [2] can depend on the transformation of axioms within the search component.
- **Heuristic vs. Search + Heuristic:** Compare the effects of applying the transformation to only heuristic, and both, heuristic and search, to isolate the impact of axiom evaluation in heuristics from its impact during state-space search.
- **Performance impact:** Evaluate the influence of the transformation on *runtime* and *memory consumption* across a diverse benchmark suite.

1.3 Thesis Structure

This thesis is structured as follows:

- **Chapter 2** introduces the theoretical background of this thesis. It explains planning tasks and axioms in FDR as well as the Tseitin transformation.
- **Chapter 3** presents related work.
- **Chapter 4** explains the concrete implementation of the Tseitin transformation in the *Fast Downward* planning system.
- **Chapter 5** describes the methodology chosen to perform the empirical analysis, including benchmark selection and measurement procedures and also presents and discusses the results of the runtime and memory analyses and compares the transformed with the non-transformed variant on selected search algorithms and heuristics.
- **Chapter 6** summarises the most important results and provides an outlook on possible further developments and open research questions.

2 Background

2.1 Planning tasks in FDR

We follow the definition of finite-domain representation (FDR) tasks by Helmert [7].

Definition 2.1 (FDR planning task)**.** *An* FDR planning task *is a 5-tuple*

$$\Pi = \langle V, s_0, s_g, A, O \rangle,$$

where

- *V is a finite set of state variables, each $v \in V$ with a finite domain D_v. Here we distinguish between state variables, which can be affected by operators (*fluent variables*) $v_f \in V_f$ and state variables computed by evaluation axioms (*derived variables*) $v_d \in V_d$ so that $V = V_d \cup V_f$. The domain of derived variables contain the* default value $\perp$ *(unsatisfied).*

  We will use the general definition of a state *s in FDR, as a total function*

  $$s : V \to \bigcup_{v \in V} D_v$$

  such that

  $$s(v) \in D_v \quad \textit{for all } v \in V.$$

  A partial assignment thus specifies values only for a subset of the state variables. More generally, a partial variable assignment *p is a function*

  $$p : V' \to \bigcup_{v \in V'} D_v$$

  defined on some subset $V' \subseteq V$, with $p(v) \in D_v$ for all $v \in V'$.

- *s_0 is a state over V, defined on all $v_f \in V$, but not on any $v_d \in V$, called the initial state.*

- *s_g is a partial variable assignment over V, called the goal.*

- *A is a finite, layered set of axioms over V. Axioms are defined as* cond $\to v := d$, *where* cond *is a partial assignment (the body), v is a derived variable (the affected variable), and $d \in D_v$ is the derived value. The set A is partitioned into a totally ordered sequence of axiom layers $A_1 \prec \cdots \prec A_k$ satisfying the* layering property *(see definition 2.2)*
- *O is a finite set of operators. An operator is a pair $\langle \text{pre}, \text{eff} \rangle$, where* pre *is a partial assignment over V (the precondition), and* eff *is a set of conditional effects* cond $\to$ $v := d$ *with* cond *a partial assignment over V, $v \in V_f$, and $d \in D_v$.*

Definition 2.2 (Layering Property [7])**.** *A set of axioms A can be partitioned into layers $A_1 \prec \cdots \prec A_k$ if the following conditions hold:*

- *Within one layer, two axioms with the same head variable must not assign different values to it.*
- *If a variable appears as a head in some layer A_i, it must not appear with a different value in the body of any axiom within the same layer.*

The layering property ensures that axioms can be evaluated layer by layer until a fixed point is reached, guaranteeing termination and determinism of axiom evaluation.

Definition 2.3 (Extended state)**.** *Let s be a state over V_f. The corresponding* extended state *$s^\star$ is obtained as follows:*

1. *Initialise all derived variables $v \in V_d$ with the default value $\perp$.*
2. *For each axiom layer A_i in order $A_1 \prec \cdots \prec A_k$:*
   a) *While there exists an axiom in A_i whose body is satisfied in the current assignment and whose head variable currently has value $\perp$, update that variable to the value specified by the axiom head.*

By the layering property of axioms, this process terminates after finitely many steps and results in a unique extended state $s^\star$.

Definition 2.4 (Atom)**.** *Let V be the set of variables of a planning task, where each $v \in V$ has a finite domain D_v. An* atom *is an expression of the form*

$$(v = d),$$

where $v \in V$ and $d \in D_v$.

For the remainder of this thesis, we adopt a slightly simplified notation for axioms in order to improve readability.

Definition 2.5 (FDR Axiom)**.** *Let V be the set of variables of a planning task with $V = V_d \cup V_f$ where V_d is the set of derived variables and V_f is the set of fluent variables. An* axiom *is a rule of the form*

$$(v_1 = d_1 \wedge v_2 = d_2 \wedge \cdots \wedge v_k = d_k) \;\Rightarrow\; (u := e)$$

where each $v_i \in V$ and $u \in V_d$ is a variable, and $d_i \in D_{v_i}$, $e \in D_u$. The conjunction on the left-hand side is called the body, *the assignment on the right-hand side the* head.

For convenience, we map each assignment $(v = d)$ to a propositional atom p and $u := d$ to q. Thus, an axiom can equivalently be written as

$$(p_1 \wedge p_2 \wedge \cdots \wedge p_k) \;\Rightarrow\; q$$

where each p_i and q denotes a variable–value assignment.

2.2 Tseitin Transformation of Axioms

The *Tseitin transformation* is a widely used technique to convert arbitrary propositional formulas into a conjunctive normal form (CNF) that is *equisatisfiable* to the original formula. It was first introduced by Tseitin [19] and later refined by Plaisted and Greenbaum [14]. The central idea is to avoid the exponential blow-up of a direct CNF conversion by introducing fresh auxiliary variables for subformulas.

Definition 2.6 (Tseitin Transformation [14, 19]). *Let φ be a propositional formula. The Tseitin transformation constructs a CNF formula ψ together with a set of fresh auxiliary variables Y, such that*

$$\varphi \text{ is satisfiable} \quad \Leftrightarrow \quad \psi \text{ is satisfiable.}$$

For each subformula θ of φ, the transformation introduces a new variable $y_\theta \in Y$ and adds clauses enforcing $y_\theta \leftrightarrow \theta$. The final CNF ψ contains only clauses of size at most three and its size is linear in the size of φ.

Example: Consider the formula

$$\varphi = (a \wedge b) \vee c.$$

The Tseitin transformation introduces auxili¡ary variables y_1 for the subformula $(a \wedge b)$ and y_2 for the top-level formula φ. This yields the equivalences

$$y_1 \leftrightarrow (a \wedge b), \qquad y_2 \leftrightarrow (y_1 \vee c),$$

together with the constraint y_2 to enforce equivalence with φ.
Replacing the equivalences by their conjunctive normal form results in the clauses

$$\begin{aligned}
&(\neg y_1 \vee a) \wedge (\neg y_1 \vee b) \wedge (\neg a \vee \neg b \vee y_1),\\
&(\neg y_2 \vee y_1 \vee c) \wedge (\neg y_1 \vee y_2) \wedge (\neg c \vee y_2),\\
&y_2.
\end{aligned}$$

Thus the resulting formula is

$$\psi = (\neg y_1 \vee a) \wedge (\neg y_1 \vee b) \wedge (\neg a \vee \neg b \vee y_1) \wedge (\neg y_2 \vee y_1 \vee c) \wedge (\neg y_1 \vee y_2) \wedge (\neg c \vee y_2) \wedge y_2.$$

Clearly, φ is satisfiable if and only if ψ is satisfiable. However, the two formulas are not logically equivalent, since ψ contains the additional auxiliary variables y_1, y_2.
Plaisted and Greenbaum [14] observed that this is stronger than necessary for satisfiability.

Polarity-aware Plaisted–Greenbaum (PG) encoding: An occurrence of a subformula has a *polarity* (positive or negative), defined inductively: the outermost formula occurs positively; in $\neg\alpha$ the polarity of α flips; in $\alpha \wedge \beta$ and $\alpha \vee \beta$ both subformulas inherit the current polarity.
PG introduces a fresh variable y_α for a subformula α and, depending on its polarity, keeps only one implication:

$$\text{positive polarity of } \alpha: \ y_\alpha \rightarrow E(\alpha) \qquad\qquad \text{negative polarity of } \alpha: \ E(\alpha) \rightarrow y_\alpha,$$

where $E(\alpha)$ is α in terms of the proxy variables of its immediate subformulas. Finally, the top-level proxy y_φ is forced to true by the unit clause y_φ. The implications are then written as CNF clauses.

Example (PG): Let $\varphi = (a \wedge b) \vee c$ and introduce $y_1 \leftrightarrow (a \wedge b)$, $y_2 \leftrightarrow (y_1 \vee c)$ as proxies (we do not introduce proxies for literals).
Since all occurrences are positive, PG keeps

$$y_2 \rightarrow (y_1 \vee c), \qquad y_1 \rightarrow (a \wedge b), \qquad y_2.$$

In CNF this yields the clauses

$$(\neg y_2 \vee y_1 \vee c) \ \wedge \ (\neg y_1 \vee a) \ \wedge \ (\neg y_1 \vee b) \ \wedge \ y_2.$$

Thus the resulting CNF is

$$\psi \ = \ (\neg y_2 \vee y_1 \vee c) \ \wedge \ (\neg y_1 \vee a) \ \wedge \ (\neg y_1 \vee b) \ \wedge \ y_2,$$

which is equisatisfiable with φ but strictly smaller than the full Tseitin encoding.

Application to Axioms: While originally developed for propositional satisfiability, the principle of introducing auxiliary variables to break down large formulas can also be applied to axioms in planning tasks. In this thesis, we adapt the Tseitin transformation after Plaisted and Greenbaum [14] to restructure axioms whose bodies contain many conditions. Instead of encoding the axiom body as a single wide conjunction, we introduce auxiliary derived variables that represent intermediate conjunctions. This ensures that each axiom body contains at most two literals, while preserving the semantics of the original FDR task.

3 Related Work

The restructuring of axioms through a Tseitin transformation introduced in this thesis connects to several strands of research in artificial intelligence. A central reference point is the SAT and logic community, where the Tseitin transformation [19], and especially the Plaisted–Greenbaum variant [14], have long been standard techniques for converting formulas into CNF. Such transformations make it possible to handle large and complex expressions in a more compact form without altering satisfiability. This idea of replacing complex structures with auxiliary variables to simplify reasoning serves as one of the key inspirations for our work.

Within automated planning, derived predicates (originally called "axioms" in early PDDL [12]) were already part of the original *PDDL* specification but saw little practical use; they were standardised and adopted for competition benchmarks with *PDDL 2.2* [3, 8]. Derived predicates greatly increase the expressiveness of planning domains [18], but they can also complicate search and heuristic evaluation. Helmert [6, 7] describe how axioms are integrated into the planning system *Fast Downward* and formalize the layered fixed-point semantics that underlie their evaluation. This semantic framework also provides the theoretical foundation on which our approach is built.

Research on improving translations in planners has explored a variety of strategies. Rintanen [15] studied compact representations of logical dependencies through regression techniques for classical and nondeterministic planning. While not directly concerned with axioms, this work similarly illustrates how different formulations of logical dependencies can affect the efficiency of planning systems.

Another line of research integrates axioms directly into heuristic computation. Ivankovic and Haslum [10], for example, extend pattern database and abstraction heuristics to correctly account for derived predicates. Their results demonstrate that axioms can substantially affect heuristic estimates and that specialized techniques are required to incorporate them effectively. Rather than designing new heuristics, this thesis focuses on restructuring axiom bodies, which may in turn influence how existing heuristics perform.

Dedicated search techniques for planning with derived predicates have also been proposed. Gerevini et al. [4] introduced an approach based on Rule–Action Graphs combined with local search to reason more effectively in domains with axioms. Whereas their work adapts the search procedure itself, our contribution operates at the representational layer by decomposing axioms through a Tseitin-inspired transformation within the *Fast Downward* framework.

Speck et al. [17] investigates alternative ways of axiom encoding within symbolic planning. Their results demonstrate that the native axiom support in symbolic representation within optimal planners outperforms those planners, and even in planners not supporting axioms they showed that the native axiom support in symbolic representation leads to an overall benefit in classical planning domains, highlighting the practical relevance of how axioms are represented.

More recently, Borgwardt et al. [2] investigated planning with ontologies and showed that a Tseitin-style transformation of axioms can significantly increase the number of solved instances in benchmark domains. Their results suggest that restructuring axioms may have a strong practical effect. However, their work focuses on the compilation of ontology-based axioms, while our approach investigate the direct impact of applying such a transformation to FDR axioms within the search component of *Fast Downward.*

Taken together, these strands of research illustrate that axioms can be addressed at different levels of a planning system: through the internal representation of logical dependencies, by adapting heuristics, or by developing specialized search strategies. In contrast to previous work, this thesis does not introduce new heuristics or search algorithms, but instead targets the internal representation of axioms. By applying a Tseitin-inspired transformation to restructure axiom bodies, we aim to assess if such changes can influence the overall coverage of benchmark domains and how those impact runtime and memory consumption in an established planning system.

4 Implementation

This thesis proposes a *post-translation* optimisation that restructures only those axioms whose *body* contain more than two conditions. Such “wide” axioms are split into smaller, but logical equivalent axioms using a Tseitin-inspired transformation that introduces new auxiliary axioms that combines two literals in the original *body* of the previous axiom, and create a new axiom with those two literals in its *body*. This encoding is performed to break every axiom *body* down to ensure a size of at most two literals, which ensures a lower amount of checks per axiom during the evaluation.
Before we explicitly explain the details of the implementation itself, we first need to introduce some key components of *Fast Downward*, ensuring that the following sections remain understandable even for readers who are not yet familiar with the system.

4.1 *Fast Downward* Task Components

Background: PDDL and FDR: Planning problems are typically specified in the *Planning Domain Definition Language (PDDL)* [12]. PDDL provides a standardized, human-readable formalism based on first-order logic, in which actions are defined by preconditions and effects, and planning tasks consist of an initial state and a goal condition. While PDDL is well suited as an input language, it is not directly optimized for efficient search.
To this end, *Fast Downward* employs the *finite-domain representation (FDR)* [7], a propositional, variable-based encoding in which states are total assignments over finite-domain variables. All further components of the planner—including search and heuristics—are defined over this representation.

Translator (PDDL $\rightarrow$ FDR) : Note that the translator is a central component of *Fast Downward* but will not be fully covered here. We will only describe the relevant parts for this thesis. For further details, see [7].

Fast Downward uses a *Translater*, that translates a *PDDL*-Domain into a *Finite Domain Representation (FDR)*.

The output of the translator is a fully grounded, finite-domain task. This representation preserves the semantics of the original *PDDL* problem but makes it directly usable by the search and heuristic components of the planner.

AbstractTask: *AbstractTask* provides a general interface for representing a planning task in *Fast Downward*. It provides queries for Amount of Variables and Domains as well as operators and their effects, axioms, costs etc. Most of *Fast Downwards* components such as search and heuristic implement this abstract API instead of a discrete data-structure, which allows the implementation of task transformations on a grounded problem, by implementing a new type of *AbstractTask*.

RootTask: In *Fast Downward*, the currently active planning task is managed via a global pointer called the *RootTask*. This global reference always points to exactly one instance of an *AbstractTask*. All components that need to read information about the task (such as search algorithms or heuristics) do not access the *RootTask* directly, but instead use a *TaskProxy*(see 4.1).
This design makes it possible to transparently exchange the underlying task: if we want to perform a transformation, the global *RootTask* pointer is simply redirected to a new *DelegatingTask* (see 4.1) instance that wraps the original task. All other components remain unchanged, as they continue to access the task through their proxies.

DelegatingTask: The *DelegatingTask* is an adapter class that implements the *AbstractTask* interface by internally holding a pointer to another *AbstractTask*, called its *parent*. By default, every request to the *DelegatingTask* is forwarded to its parent task.
This design follows the delegation pattern and allows new functionality to be introduced without modifying the original *AbstractTask*. A *DelegatingTask* can selectively override specific methods while relying on the parent task for all others. In this way, transformations of the task description can be implemented in a clean, encapsulated manner: they change only the relevant aspects while preserving the semantics of the underlying task. Moreover, such transformations can be enabled or disabled by simply inserting or removing the corresponding *DelegatingTask* in the task hierarchy.

TaskProxy: *TaskProxy* is an access layer for client code (search/heuristics).It encapsulates an *AbstractTask* and provides convenient, type-safe access functions. The proxy itself has no semantics of its own, it only simplifies read access to the currently active task.

The following figure will sketch the workflow of *Fast Downward* without and with a transformation included.

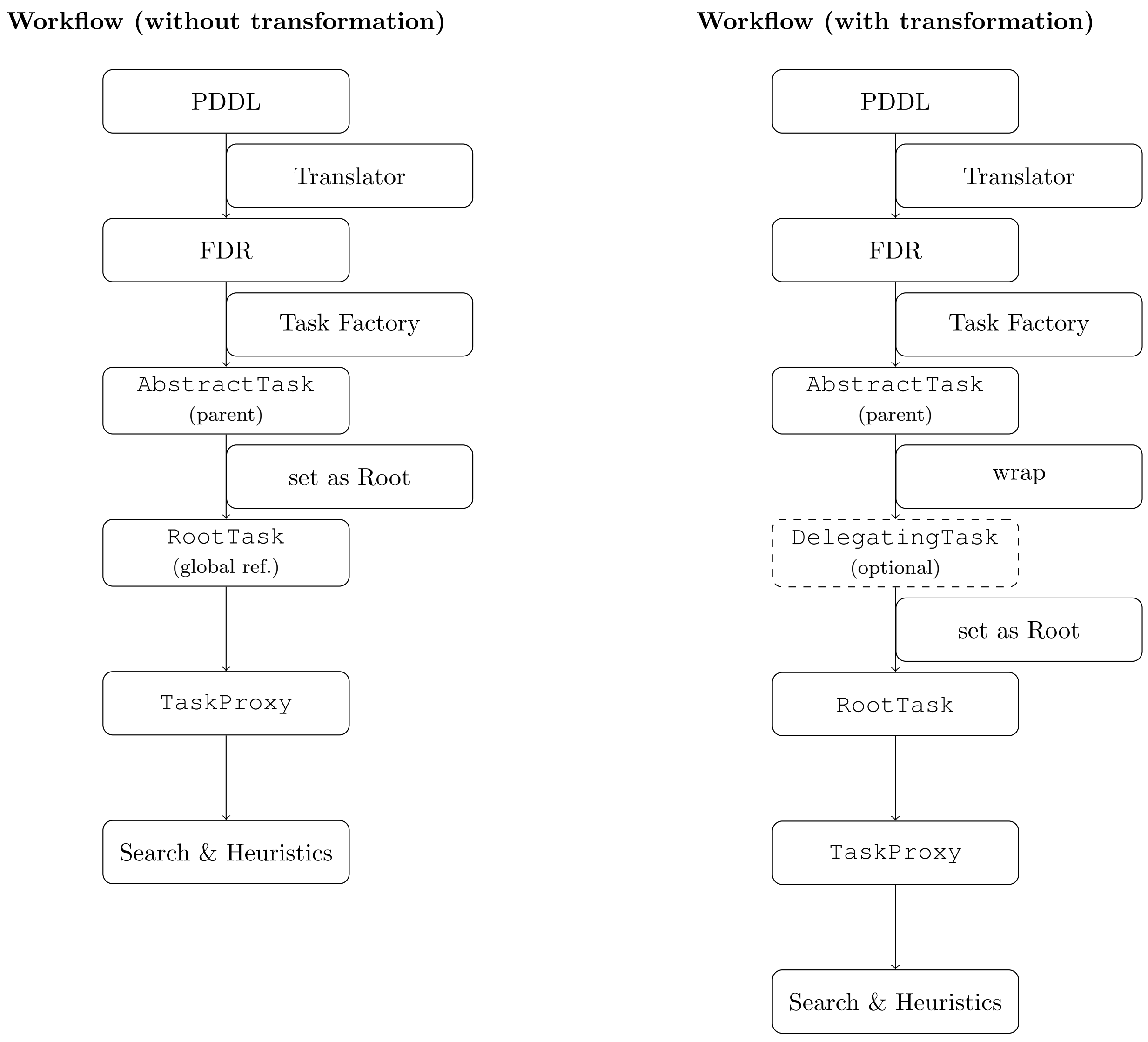


Figure 4.1: Comparison of the task workflow in *Fast Downward* without (left) and with (right) a transformation.

4.2 Axiom Transformation

Our *Tseitin-inspired axiom transformation* is applied *after Fast Downward*'s translator has created the Finite Domain Representation (FDR), and therefore ensuring no interferences with the grounding process so that the initial encoding of *Fast Downward* remains unchanged. Note that we perform the transformation only on the axioms. All other components remain unchanged. In particular, the transformation takes place between translation and the search process, affecting only the evaluation during search node expansion and heuristic computation. However, *Fast Downward* does not support such a transformation within the search by default; therefore, an interface for this transformation first had to be

implemented in the planner.
The reason behind choosing the post-translation stage for our transformation is that we have three main advantages.

1. It isolates the transformation from the grounding process allowing a simple enable and disable of the transformation without affecting other planning components.
2. It allows to compare a standard axiom-encoding evaluation with a transformed one on the same grounded problem. This is important to ensure that differences are caused by the transformation and not by a different grounded problem.
3. The interface implementation for the transformation was simple because we could introduce a new parameter for the command line which would allow us to modify the task when all necessary components for the task already exist.

The implementation of the Tseitin-Axiom transformation was integrated into the existing planning architecture of *Fast Downward* as a standalone plugin to ensure encapsulation and allows us to omit it if the transformation is not required.

The main components of the implementation are:

1. **Plugin *TseitinFeature***
   This class serves as the interface of the Transformation and enables the command-line option `transform=tseitin()`. The interface workflow is as follows:

   - Retrieve the original problem from `AbstractTask` and wrap it through `TaskProxy`.
   - Initialise and invoke the `TseitinTransformer` with this `TaskProxy` as argument.
   - The result contains a list of transformed Tseitin axioms.
   - If no auxiliary variables were created, we know that all axioms have a number of conditions of at most two, so we return the unchanged original problem.
   - Else we create a new `DelegatingTask`-object (a TseitinTask, see 3), which manages all transformed axioms and auxiliary variables.
   - This `TseitinTask` replaces the original task and is handed over to the search and heuristic.

2. **Class *TseitinTransformer***
   This class contains the core implementation of the Tseitin transformation as well as an optimisation approach.

   a) Introducing the `combination cache:` a hashmap that stores all previously created auxiliary variables in order to avoid duplicates. If a combination of literals has already been replaced in the past, the same auxiliary variable is reused. This is particularly beneficial in benchmark domains with many wide axioms, where different axioms often share parts of their *body*. Reusing auxiliary variables in such cases reduces the overall number of axioms and enables more efficient state-space search, which can lead to substantial memory savings when shared structures occur frequently.

b) For every axiom the conditions are pre-sorted by their variable to easier check if a combination of literals was seen before in the transformation process, also allowing an easier duplicate elimination. Since our transformation will systematically go through the *body* of each axiom starting from the backend, we can therefore discover if a combination of literals was seen before in an efficient manner by this pre-sorting. This approach will not discover all shared literal combinations and can be further improved.

The transformation operates as follows:

- The original set of axioms is retrieved from the `TaskProxy`.
- For each axiom $(p_1 \wedge \cdots \wedge p_k) \Rightarrow q$, where each p_i and q denotes a variable–value assignment (atom), the algorithm ensures that the body contains at most two conjuncts by a recursive encoding:
  - If $k \leq 2$, leave the axiom unchanged.
  - If $k > 2$:
    a) Order the conjuncts $p_1, \ldots, p_k$ by their underlying variable v.
    b) Let $a := p_{k-1}$ and $b := p_k$.
    c) Check the *combination cache* C:
       * If (a, b) is present , reuse the cached auxiliary atom $u_i := C(a, b)$.
       * Otherwise, create a fresh auxiliary atom u, set $C(a, b) \leftarrow u$, and add the auxiliary axiom $(a \wedge b) \Rightarrow u_i$ to the axiom set.
    d) Replace the tail $a \wedge b$ in the body by u, obtaining $(p_1 \wedge \cdots \wedge p_{k-2} \wedge u_i) \Rightarrow q$, and set $k \leftarrow k - 1$.
    e) Repeat while $k > 2$ (increase index i).
- After processing, the final axiom $(p_1 \wedge u_n \Rightarrow q)$ is added to the transformed axiom set, together with all auxiliary axioms.

- Here is a short example of the recursive encoding of an axiom:

  $(p_1 \wedge p_2 \wedge p_3 \wedge p_4) \;\Rightarrow\; q$

  Step 1: combine (p_3, p_4) into u_1 : $(p_3 \wedge p_4) \;\Rightarrow\; u_1$

  body becomes: $(p_1 \wedge p_2 \wedge u_1) \;\Rightarrow\; q$

  Step 2: combine (p_2, u_1) into u_2 : $(p_2 \wedge u_1) \;\Rightarrow\; u_2$

  Final axiom: $(p_1 \wedge u_2) \;\Rightarrow\; q$

  $$\text{List of axioms: } \begin{cases} (p_1 \wedge u_2) \Rightarrow q, \\ (p_2 \wedge u_1) \Rightarrow u_2, \\ (p_3 \wedge p_4) \Rightarrow u_1 \end{cases}$$

3. **Class *TseitinTask***
   `TseitinTask` implements `DelegatingTask` and references the original `AbstractTask` of *Fast Downward*. The original operators stay the same but since `TseitinTask` gets a new list of axioms and conditions, it is important to cover these changes in the original task. Those features are:

   - **Number of variables:** since we add new axioms and auxiliary variables, we need to adjust the total number of variables we have in our task.
   - **Domain size:** our new auxiliary variables and in particular all axioms can either be true or false, and therefore have a domain size of two.
   - **Axiom Layer:** Since the axiom layer indicates at what point in the reasoning process a variable can be derived, the layer for auxiliary variables is determined as the maximum axiom layer of all its condition variables. This is needed since the axioms *body* can have interdependences relationships, and cannot be calculated at an arbitrary time and therefore ensures that an auxiliary variable can only become true once all of its defining conditions are available.

     Theorem 4.1 (Layering from the max-equations)**.** *Let $V = V_f \cup V_d$ be the variables of the parent task and $L : V \to \mathbb{N}$ a valid variable-layer assignment whose induced axiom layering satisfies the layering property [7]. Let V_{aux} and A_{aux} be the auxiliary variables and axioms introduced by the Tseitin grouping of axiom bodies; set $A := A_{\text{par}} \cup A_{\text{aux}}$.*
     Define $L' : V \cup V_{\text{aux}} \to \mathbb{N}$ as the pointwise least solution of

     $$\begin{aligned} &L'(v) = L(v) \quad (v \in V), \\ &L'(u) = \max\{\, L'(w) \mid (w = \cdot) \\ &\textit{occurs in the body of the unique auxiliary axiom for } u \,\} \quad (u \in V_{\text{aux}}). \end{aligned} \tag{E}$$

     Let $L_{\max} := \max\{\, L'(y) \mid y \in V \cup V_{\text{aux}} \,\}$. For $i \in \{1, \ldots, L_{\max}\}$ define axiom layers by head layer

     $$\mathcal{A}_i \;=\; \{\, B \Rightarrow (x := d) \in A \mid L'(x) = i \,\},$$

and order them as $\mathcal{A}_1 \prec \mathcal{A}_2 \prec \cdots \prec \mathcal{A}_{L_{\max}}$. Then the sequence $(\mathcal{A}_1 \prec \cdots \prec \mathcal{A}_{L_{\max}})$ satisfies the layering property. Consequently, layer-by-layer evaluation until a fixpoint is reached terminates and yields a unique result.

Proof. Let $\mathcal{A}_i$ be as defined. We verify the two conditions introduced by Helmert [7].

(1) No conflicting heads within a layer. For parent heads $x \in V$, $L'(x) = L(x)$, so the parent layering already forbids two axioms in the same layer assigning different values to x. For auxiliary heads $u \in V_{\text{aux}}$, by construction there is exactly one auxiliary axiom with head u (uniqueness ensured by the combination cache), so conflicts are impossible.

(2) If a variable appears as a head in layer i, it does not occur with a different value in any body of the same layer. For parent variables this is inherited from the parent layering and is not altered by Tseitin grouping, which only introduces proxies but does not change variable–value pairs in bodies. Auxiliary variables appear in bodies only with their distinguished head value, hence cannot occur with a different value in the same layer.

It remains to note stratification: for any axiom $B \Rightarrow (x := d) \in \mathcal{A}_i$ we have by (E)

$$L'(x) \;=\; \max\{\, L'(w) \mid (w = \cdot) \in B \,\} \;=\; i,$$

thus $L'(w) \leq i$ for all variables w in the body. Therefore, when evaluating layer i, all prerequisites are already fixed by layers $< i$ or evolve monotonically within layer i. Since heads switch at most once (from $\perp$ to their target value), the intra-layer fixpoint terminates and is deterministic, and with finitely many layers the overall evaluation terminates and is unique. □

5 Evaluation

This chapter focuses on the methodology used to perform the analysis as well as presenting the results over the analysis setup.

5.1 Benchmark Suites

The empirical analysis uses a wide area of benchmarks to ensure robustnes of the results, containing benchmarks with few axioms and “small” *body* as well as complex axioms with “wide” *body* and many interdependences. All domains are taken from the benchmark repository of [13][1], which collects domains from the International Planning Competition (IPC) [9] and includes additional domains by Ivankovic et al. [11] (e.g., sokoban-axioms).
Since we are interested into domains with axioms we took following domains:

sokoban-axioms, psr-middle-noce, psr-middle, optical-telegraphs, philosophers, acc-cc2-ghosh-etal, doorexample-broken-ghosh-etal, doorexample-fixed-ghosh-etal, grid-axioms, miconic-axioms, trapping_game, collab-and-comm-kg, muddy-child-kg, muddy-children-kg, sum-kg, wordrooms-kg

We excluded *grid-cc2-ghosh-etal* because it contains an undeclared object, and *mincut* because it relies on *object fluents*, which are not supported by our planner. These domains were therefore not usable in our experiments. In addition, we had to interrupt some runs on other benchmarks due to a possible error in the validation process, which caused the plan validation to run for more than 24 hours without producing a result.

5.2 Measurement procedures

For the measurement procedure we will focus on 4 main components:

1. *heuristic value of the initial state (initial h value):* For each run, we compare the heuristic value of the initial state with and without the transformation. Since the

[1] https://github.com/dosydon/axiom_benchmarks

transformation preserves the semantics of the planning task, this value is expected to remain unchanged. Any deviation would indicate that the heuristic computation is sensitive to structural differences in the axiom encoding.

2. *coverage:* Since the work of Borgwardt et al. [2] showed a significant influence of the Tseitin transformation in terms of the number of solved instances in benchmark domains, we want to compare if the Tseitin-inspired transformation on FDR axioms has a similar influence.

3. *memory:* We measure the peak memory consumption reported by *Fast Downward*, which reflects the maximum amount of main memory used during translation, heuristic evaluation, and search. Since our transformation reduces the number of conditions in axiom *bodys* but introduces additional axioms, we are interested in the influence of this trade-off

4. *search time:* We record the search runtime from search start to search termination, and compare these values to assess whether the transformation has a measurable impact on performance.

5.3 Evaluation Setup

For the evaluation of our work we will analyse the impact on the above named criteria over the following setup:

Similar to Borgwardt et al. [2] we used Downward Lab [16] to conduct experiments with the *Fast Downward* planning system [6] on "Hardware" with a time limit of 30 minutes and a memory limit of 3 GiB per task. To stay comparable, we also adjust our search and heuristic strategies on that.

Search: We will use the eager greedy search algorithm (just as Borgwardt et al. [2]). Greedy best-first search expands nodes in order of their heuristic value. In the eager variant, all successors of an expanded node are generated and evaluated immediately.

Heuristic: We will also cover over every search two different heuristic:

1. FF-heuristic, also used by Borgwardt et al. [2]
2. Additive Heuristic

To see if there are different influences on different heuristics, that support axioms in *Fast Downward*
For completeness, we will briefly recall how those two heuristics work:

- The additive heuristic [1] estimates the cost of achieving the goal by summing the costs of achieving individual subgoals independently.
- The FF heuristic [8] approximates goal distance by extracting a relaxed plan from the delete-relaxed task.

Both heuristics are supported by *Fast Downward* and can take axioms into account.

Heuristic without search: We will also observe the differences if we only influence the heuristic but not the search itself.

Axiom Evaluation and Default Value Handling: *Fast* Downward provides two options for handling default values of derived variables, which are relevant for the evaluation of heuristics:

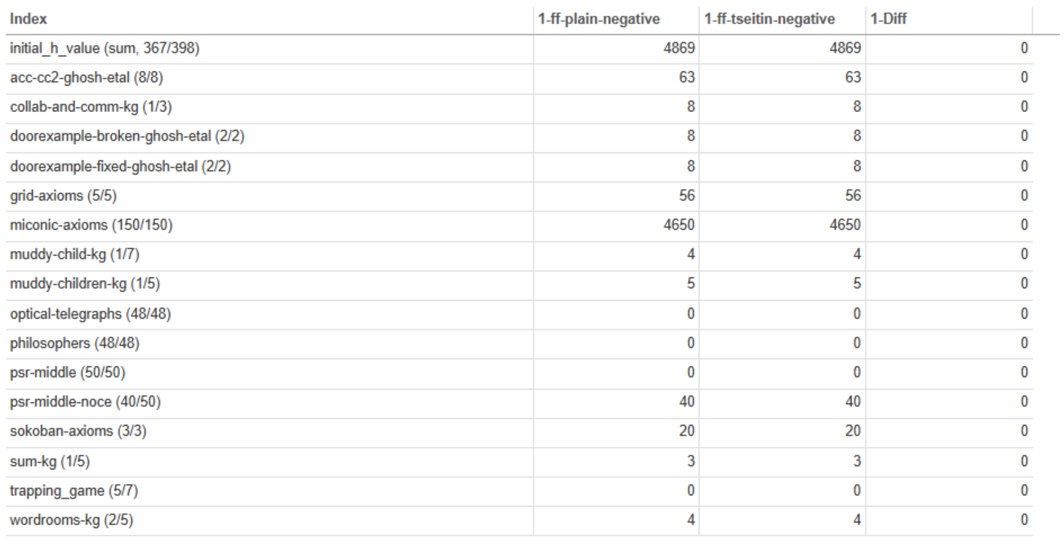
- **Approximate Negative:** Every derived variable can always take its default value unconditionally ($\neg v \leftarrow \top$). This avoids combinatorial blow-ups, but can lead to overly optimistic heuristic values, since the heuristic assumes that derived conditions can be "turned off" at no cost (cost=0).
- **Approximate Negative Cycles:** For acyclic dependencies, exact default-value axioms are derived by transforming the original axioms into CNF and then to DNF. For cyclic dependencies, the system falls back to the trivial approximation. This yields more accurate heuristics in the acyclic case, but risks a combinatorial explosion in the presence of large formulas.

Since our Tseitin transformation changes the structure of axiom bodies, it may interact differently with these two modes. We therefore evaluate our approach under both configurations.

5.4 Results

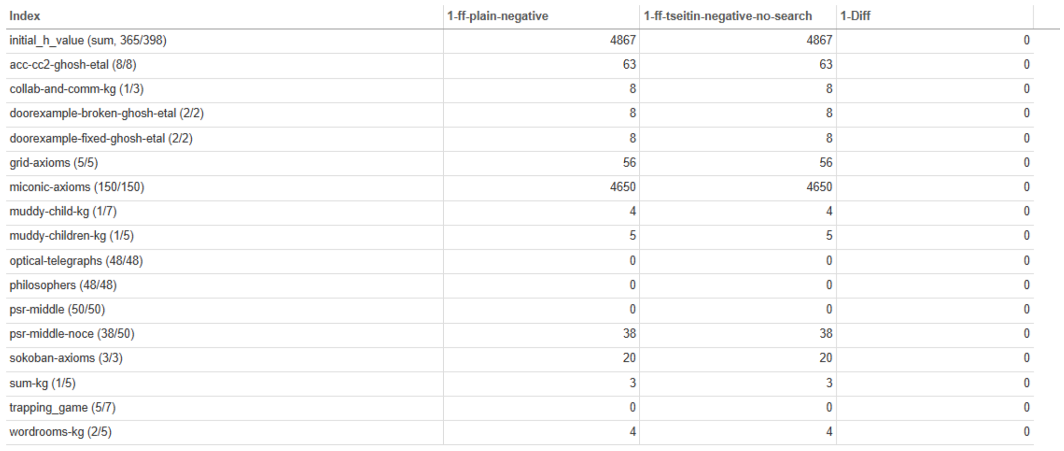
5.4.1 Initial Heuristic Values

Firstly we will observe the initial heuristic values over the FF heuristic:



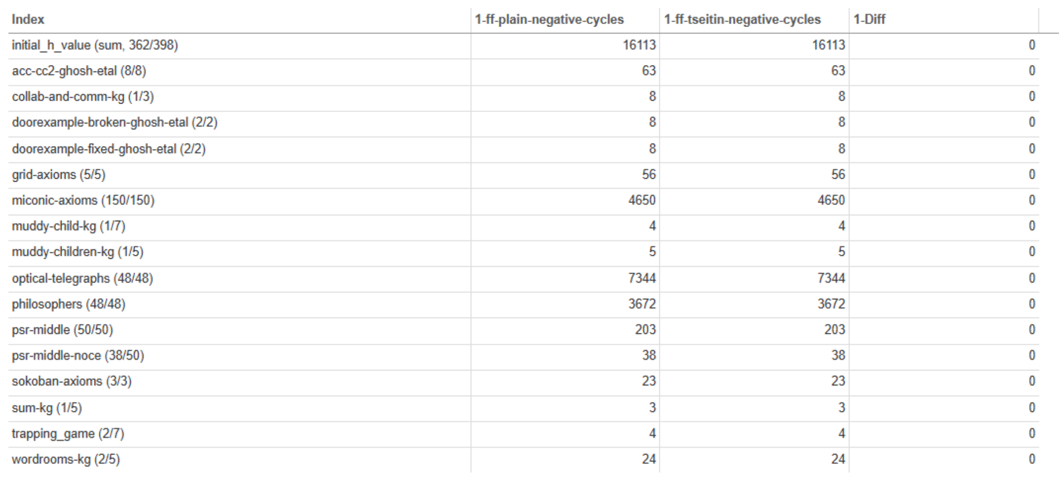
Index	1-ff-plain-negative	1-ff-tseitin-negative	1-Diff
initial_h_value (sum, 367/398)	4869	4869	0
acc-cc2-ghosh-etal (8/8)	63	63	0
collab-and-comm-kg (1/3)	8	8	0
doorexample-broken-ghosh-etal (2/2)	8	8	0
doorexample-fixed-ghosh-etal (2/2)	8	8	0
grid-axioms (5/5)	56	56	0
miconic-axioms (150/150)	4650	4650	0
muddy-child-kg (1/7)	4	4	0
muddy-children-kg (1/5)	5	5	0
optical-telegraphs (48/48)	0	0	0
philosophers (48/48)	0	0	0
psr-middle (50/50)	0	0	0
psr-middle-noce (40/50)	40	40	0
sokoban-axioms (3/3)	20	20	0
sum-kg (1/5)	3	3	0
trapping_game (5/7)	0	0	0
wordrooms-kg (2/5)	4	4	0

Figure 5.1: FF, negative



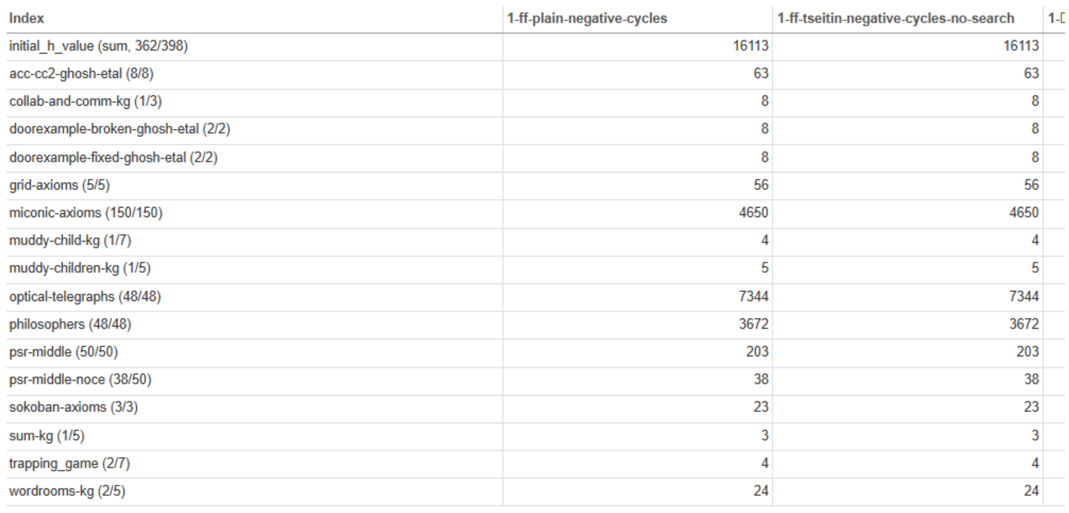
Index	1-ff-plain-negative	1-ff-tseitin-negative-no-search	1-Diff
initial_h_value (sum, 365/398)	4867	4867	0
acc-cc2-ghosh-etal (8/8)	63	63	0
collab-and-comm-kg (1/3)	8	8	0
doorexample-broken-ghosh-etal (2/2)	8	8	0
doorexample-fixed-ghosh-etal (2/2)	8	8	0
grid-axioms (5/5)	56	56	0
miconic-axioms (150/150)	4650	4650	0
muddy-child-kg (1/7)	4	4	0
muddy-children-kg (1/5)	5	5	0
optical-telegraphs (48/48)	0	0	0
philosophers (48/48)	0	0	0
psr-middle (50/50)	0	0	0
psr-middle-noce (38/50)	38	38	0
sokoban-axioms (3/3)	20	20	0
sum-kg (1/5)	3	3	0
trapping_game (5/7)	0	0	0
wordrooms-kg (2/5)	4	4	0

Figure 5.2: FF, negative, no search



Index	1-ff-plain-negative-cycles	1-ff-tseitin-negative-cycles	1-Diff
initial_h_value (sum, 362/398)	16113	16113	0
acc-cc2-ghosh-etal (8/8)	63	63	0
collab-and-comm-kg (1/3)	8	8	0
doorexample-broken-ghosh-etal (2/2)	8	8	0
doorexample-fixed-ghosh-etal (2/2)	8	8	0
grid-axioms (5/5)	56	56	0
miconic-axioms (150/150)	4650	4650	0
muddy-child-kg (1/7)	4	4	0
muddy-children-kg (1/5)	5	5	0
optical-telegraphs (48/48)	7344	7344	0
philosophers (48/48)	3672	3672	0
psr-middle (50/50)	203	203	0
psr-middle-noce (38/50)	38	38	0
sokoban-axioms (3/3)	23	23	0
sum-kg (1/5)	3	3	0
trapping_game (2/7)	4	4	0
wordrooms-kg (2/5)	24	24	0

Figure 5.3: FF, negative cycles



Index	1-ff-plain-negative-cycles	1-ff-tseitin-negative-cycles-no-search	1-
initial_h_value (sum, 362/398)	16113	16113	
acc-cc2-ghosh-etal (8/8)	63	63	
collab-and-comm-kg (1/3)	8	8	
doorexample-broken-ghosh-etal (2/2)	8	8	
doorexample-fixed-ghosh-etal (2/2)	8	8	
grid-axioms (5/5)	56	56	
miconic-axioms (150/150)	4650	4650	
muddy-child-kg (1/7)	4	4	
muddy-children-kg (1/5)	5	5	
optical-telegraphs (48/48)	7344	7344	
philosophers (48/48)	3672	3672	
psr-middle (50/50)	203	203	
psr-middle-noce (38/50)	38	38	
sokoban-axioms (3/3)	23	23	
sum-kg (1/5)	3	3	
trapping_game (2/7)	4	4	
wordrooms-kg (2/5)	24	24	

Figure 5.4: FF, negative cycles, no search

As expected, the initial heuristic values of the FF heuristic are identical in all experiments when comparing the plain encoding of axioms with the Tseitin transformed encoding. This demonstrates that the transformation does not change the semantics of the task.
The same result also holds for the *additive* heuristic, which we omit here, since the FF heuristic already provides sufficient evidence for this statement given that we use the same benchmark domains.

5.4.2 Coverage

In this section we will compare the coverage between benchmark domains and will analyse if and in what amount a difference between transformed and untransformed axioms can be observed. Note that we will not show every benchmark domain below but focus on the domains we think are necessary to show.

- FF_1 = ff plain negative
- FF_2 = ff tseitin negative
- FF_3 = ff tseitin negative no search
- FF_4 = ff plain negative cycles
- FF_5 = ff tseitin negative cycles
- FF_6 = ff tseitin negative cycles no search

- Add_1 = add plain negative
- ADd_2 = add tseitin negative
- Add_3 = add tseitin negative no search
- Add_4 = add plain negative cycles
- Add_5 = add tseitin negative cycles
- Add_6 = add tseitin negative cycles no search

Benchmark	FF_1	FF_2	FF_3	FF_4	FF_5	FF_6
grid-axioms	5	5	5	5	5	5
philosophers	5	5	5	48	48	48
psr-middle	44	44	44	44	44	44
psr-middle-noce	33	33	31	31	**33**	**32**
optical-telegraphs	2	2	2	4	4	4
tapping-game	5	5	5	2	2	2

Table 5.1: Results coverage of benchmarks

Benchmark	Add_1	Add_2	Add_3	Add_4	Add_5	Add_6
grid-axioms	5	5	5	5	0	5
philosophers	5	5	5	48	0	48
psr-middle	50	50	44	44	0	44
psr-middle-noce	34	**35**	34	33	0	**35**
optical-telegraphs	3	3	2	7	0	7
tapping-game	5	5	5	2	0	2

Table 5.2: Results coverage of benchmarks

Although most of the benchmark domain coverage remained the same, we can see that some benchmark domains were positively influenced. For example, the *ff tseitin negative cycles* variant has a positive influence compared to the plain version *ff plain negative cycles*, and even on the heuristic transformation a difference can be observed. All of those differences occurred due to interrupted runs within the experiment. In conclusion, we can say that the influence reported by [2] is **not** observable in a Tseitin-style transformation within the FDR axioms.

5.4.3 Performance influence

In this section we will have a look on the performance influence our transformation has, by analysing the time needed for search and the overall memory consumption. For this section we will use scatter Plots to compare the plain encoding with a tseitin encoding. Each caption specifies the exact experimental setup in the format:
heuristic used, approximation mode used, search + heuristic transformation(empty) or only

heuristic ("no search").
For example, the caption *"FF, negative cycles, no search"* refers to FF heuristic, approximate negative cycles, and the transformation applied only heuristic.
All figures are plotted on a log–log scale. The *x-axis* represents the baseline (no transformation), and the *y-axis* represents our Tseitin-based transformation. A reference line $x = y$ is included: points below this line indicate cases where the baseline value is larger than the transformed value, and points above the line indicate the opposite.

5.4.3.1 Search Time

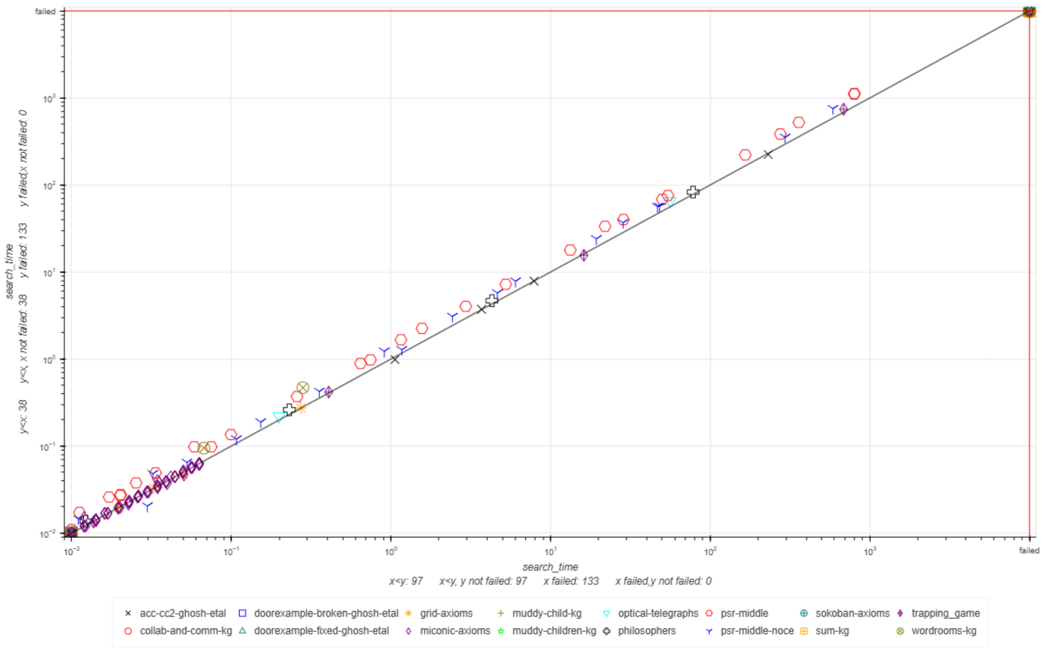


Figure 5.5: figure

FF, negative

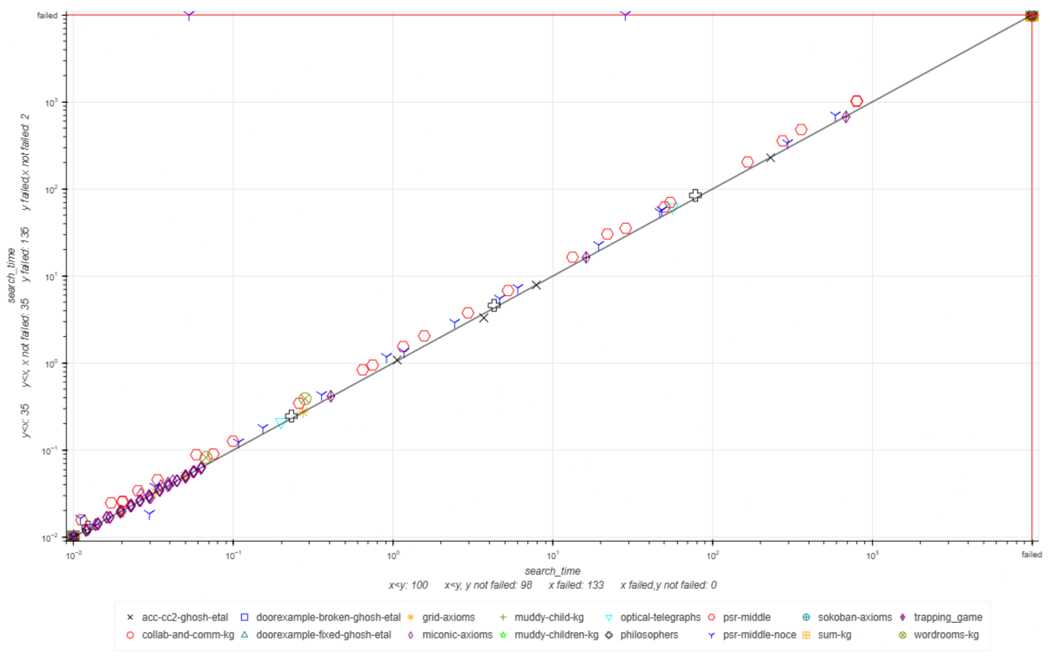


Figure 5.6: figure

FF, negative, no search

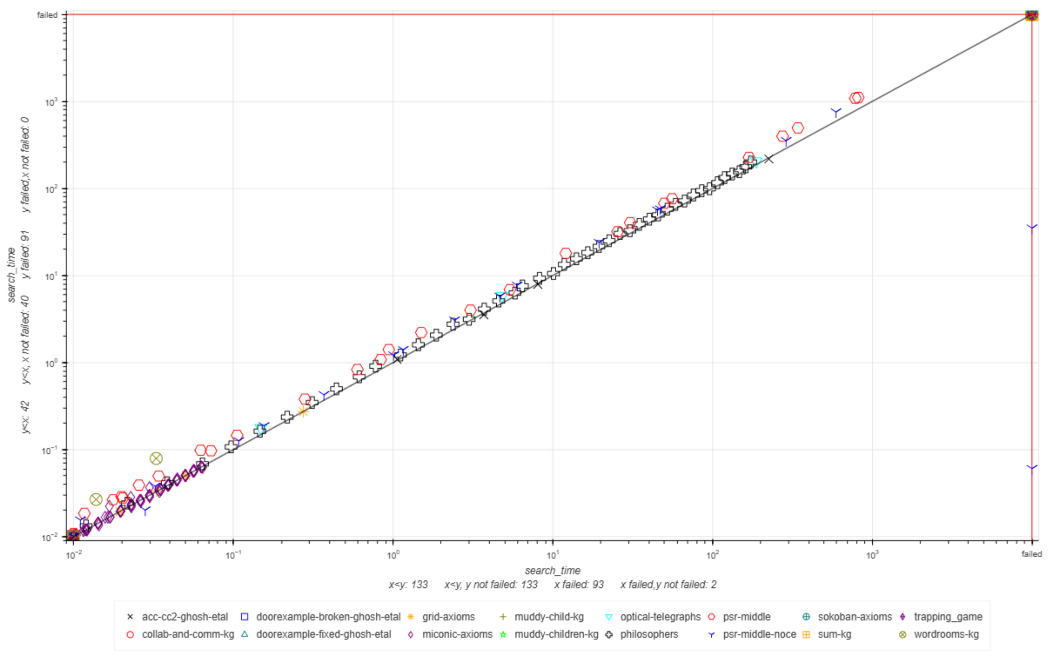


Figure 5.7: figure

FF, negative cycles

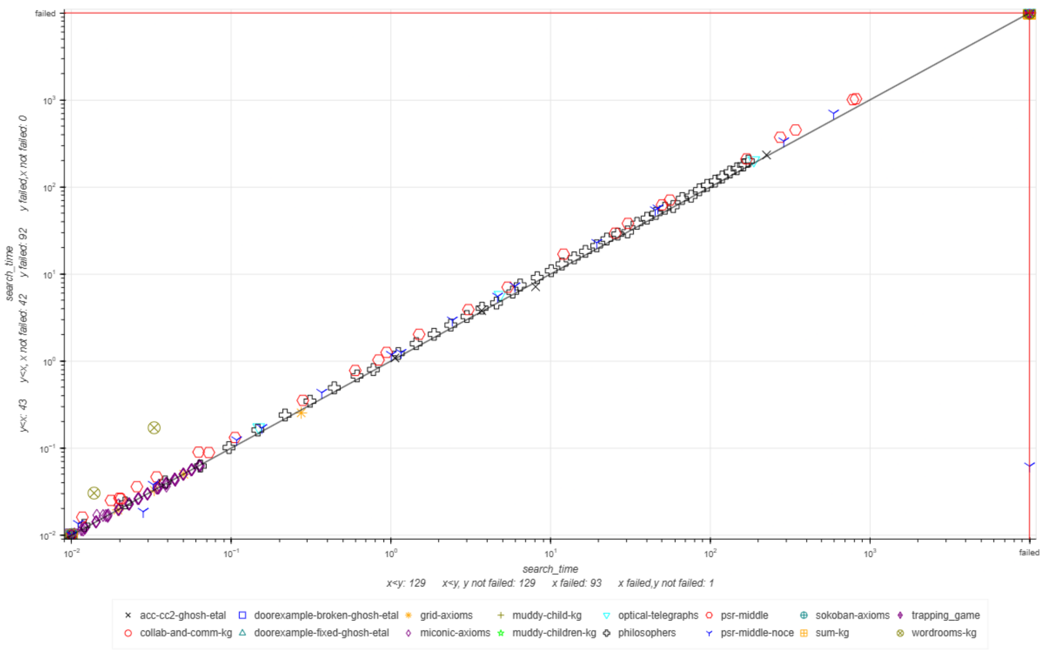


Figure 5.8: figure

FF, negative cycles, no search

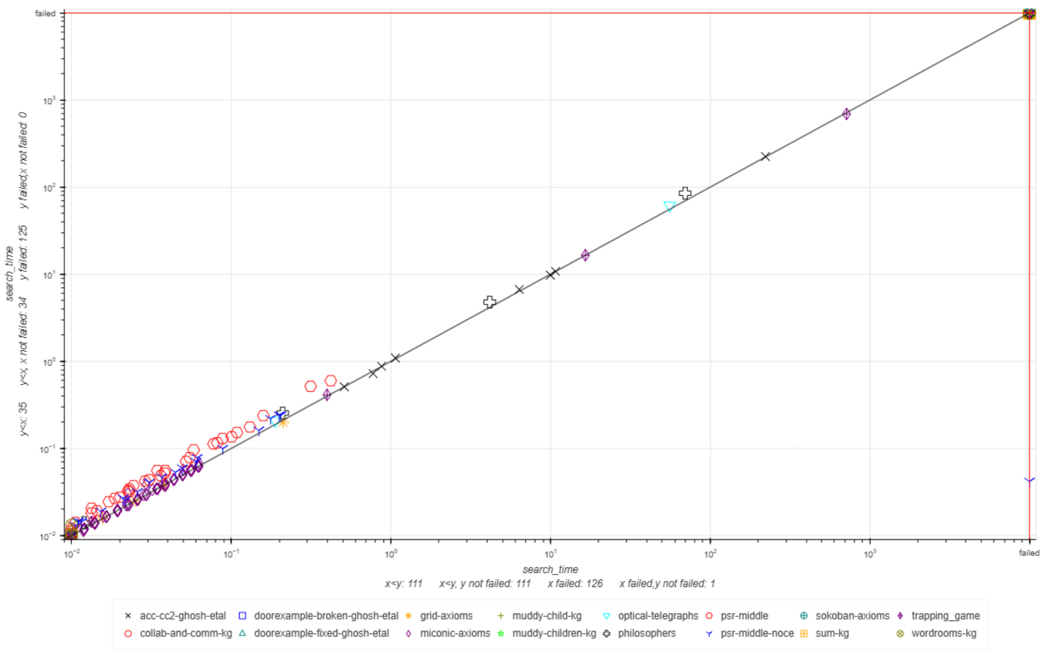


Figure 5.9: figure

add, negative

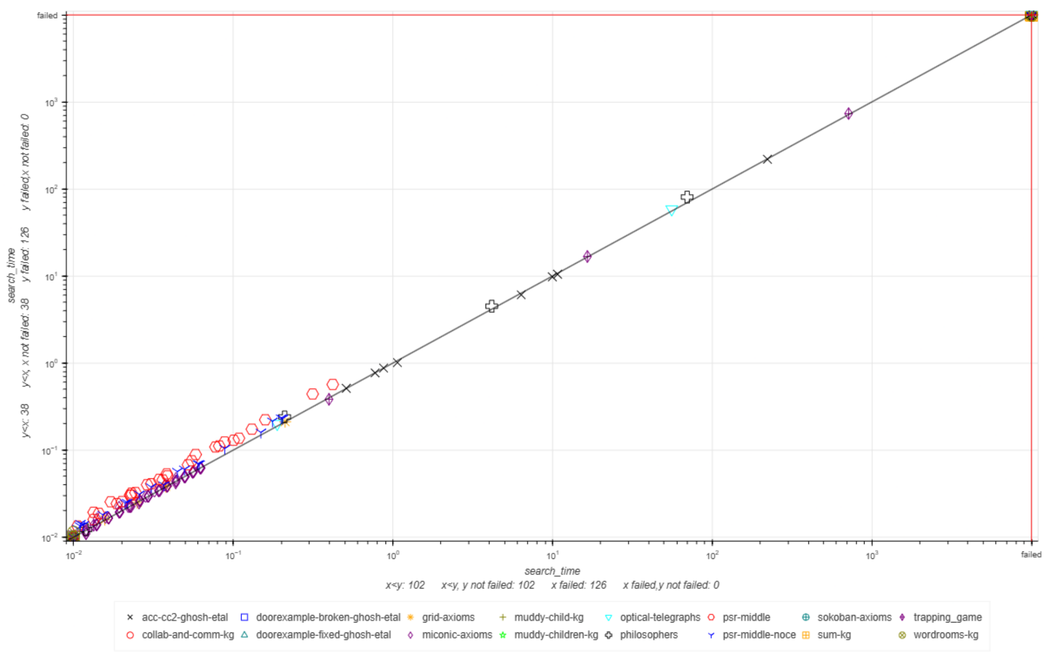


Figure 5.10: figure

add, negative, no search

`results/t add tseitin negative cycles.png`

Figure 5.11: figure

add, negative cycles

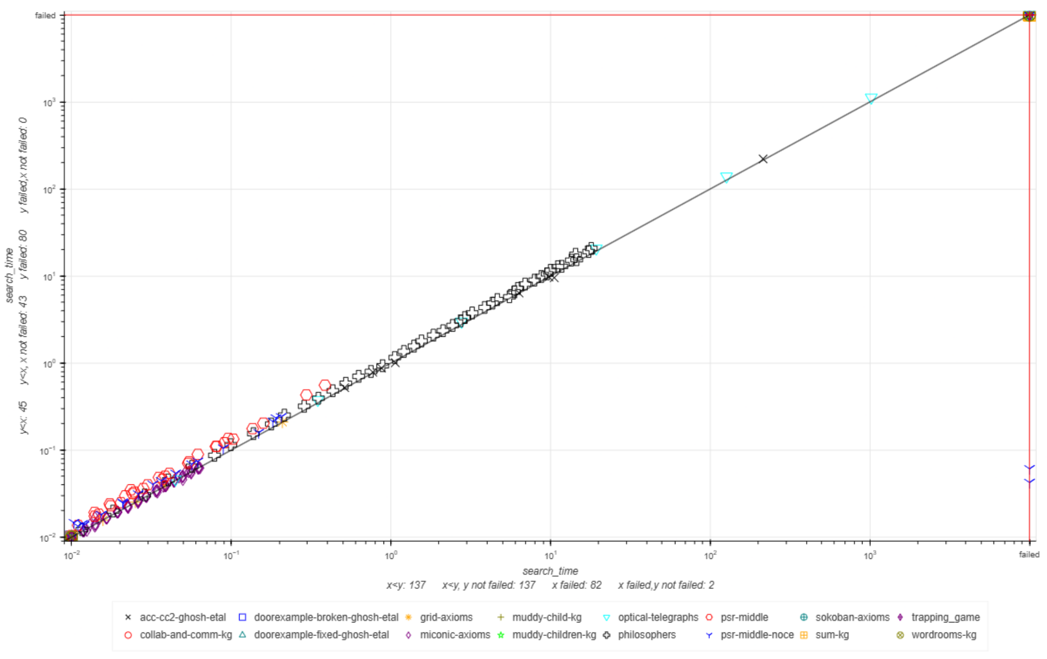


Figure 5.12: figure

add, negative cycles, no search

As we can see, the total runtime of our transformation is in most cases slightly higher than that of the untransformed computation. There are a few instances where the transformed runs are marginally faster, but these “speed-ups” are negligible and can likely be attributed to normal fluctuations in computation time. We did not observe any significant improvements that would indicate a genuine performance gain. On average, the transformation results in a small slowdown, although the effect is minor.

5.4.3.2 Memory

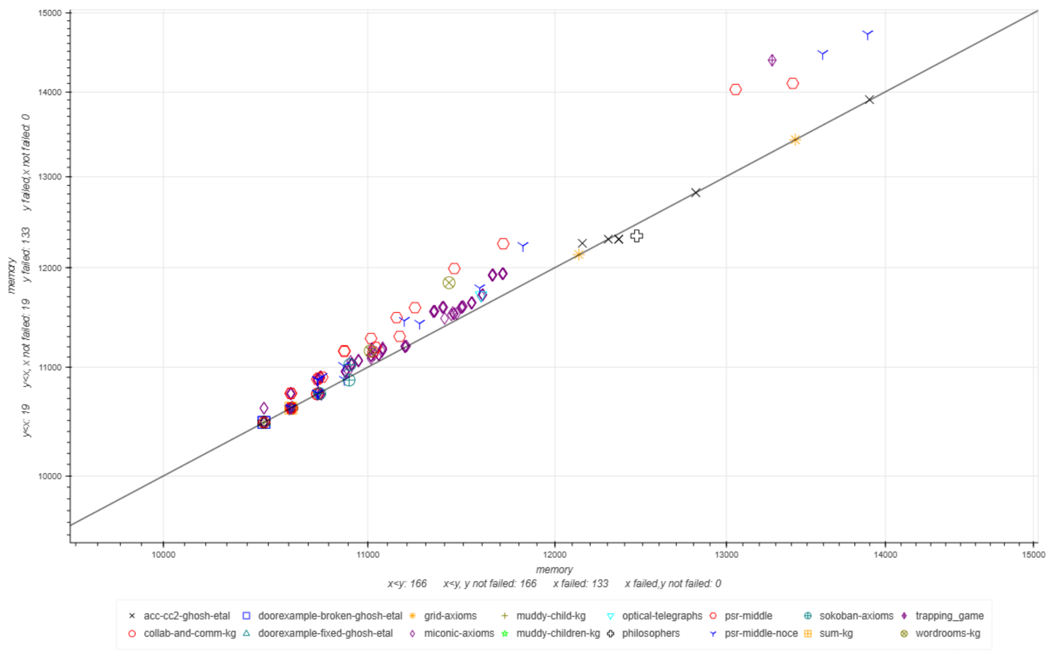


Figure 5.13: figure

FF, negative

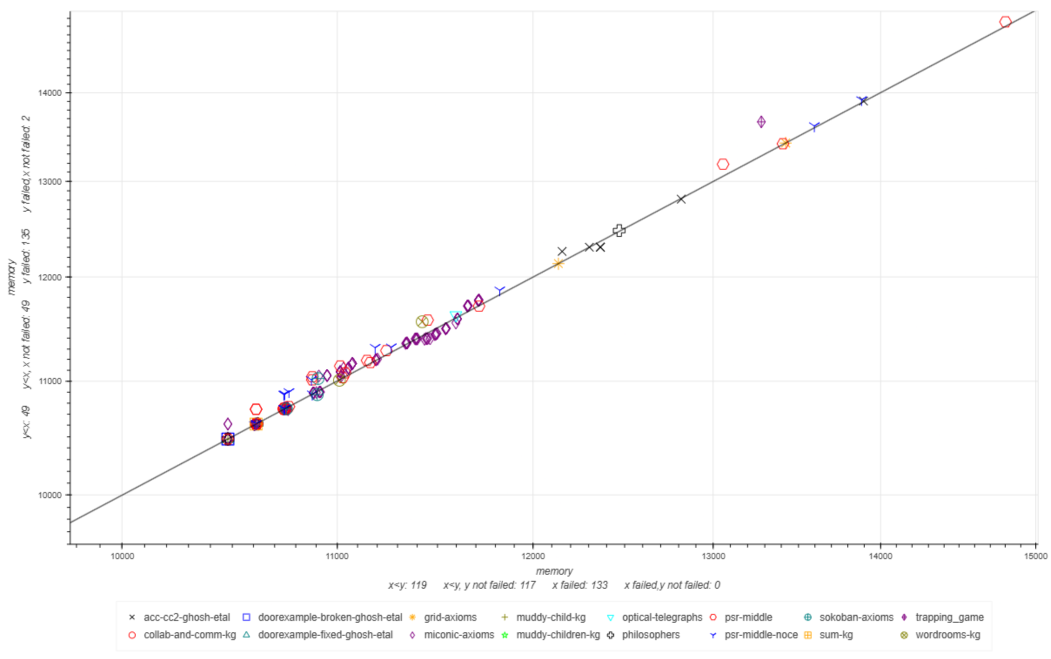


Figure 5.14: figure

FF, negative, no search

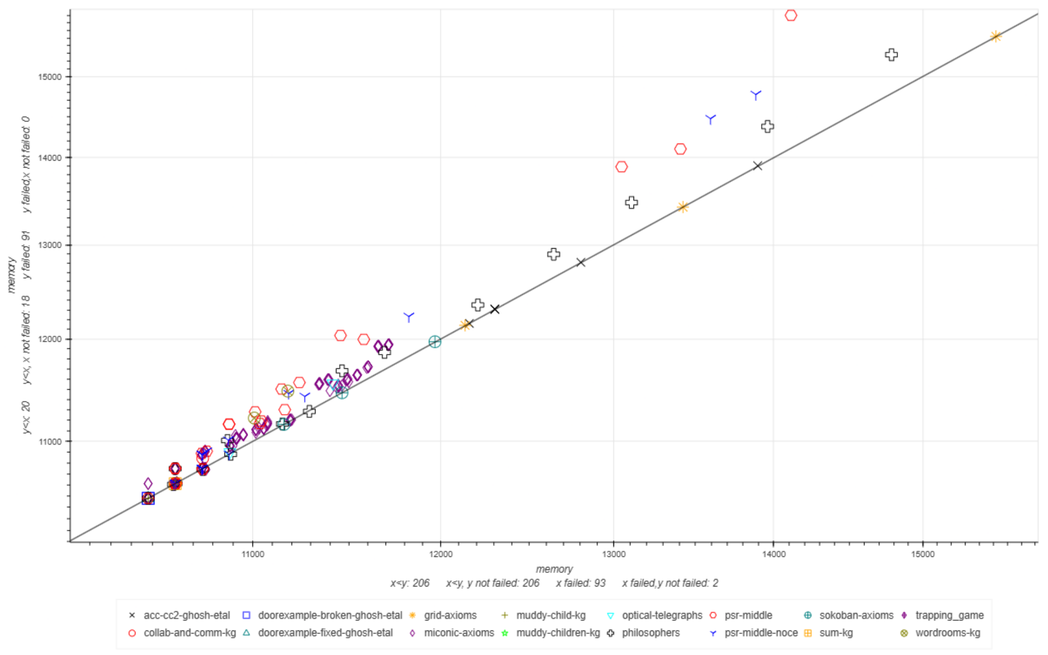


Figure 5.15: figure

FF, negative cycles

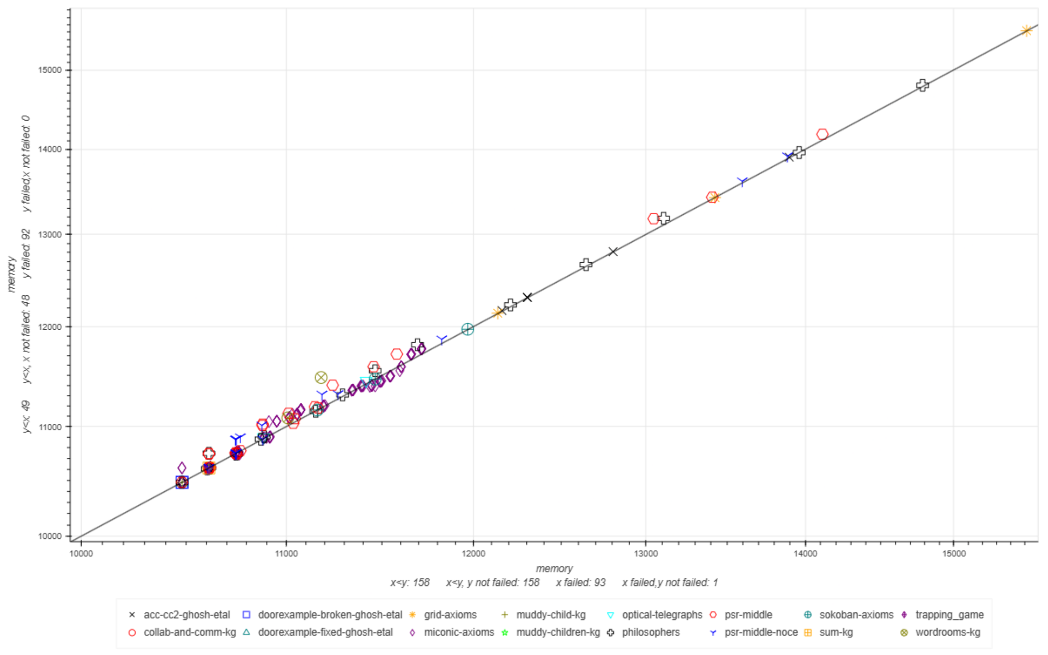


Figure 5.16: figure

FF, negative cycles, no search

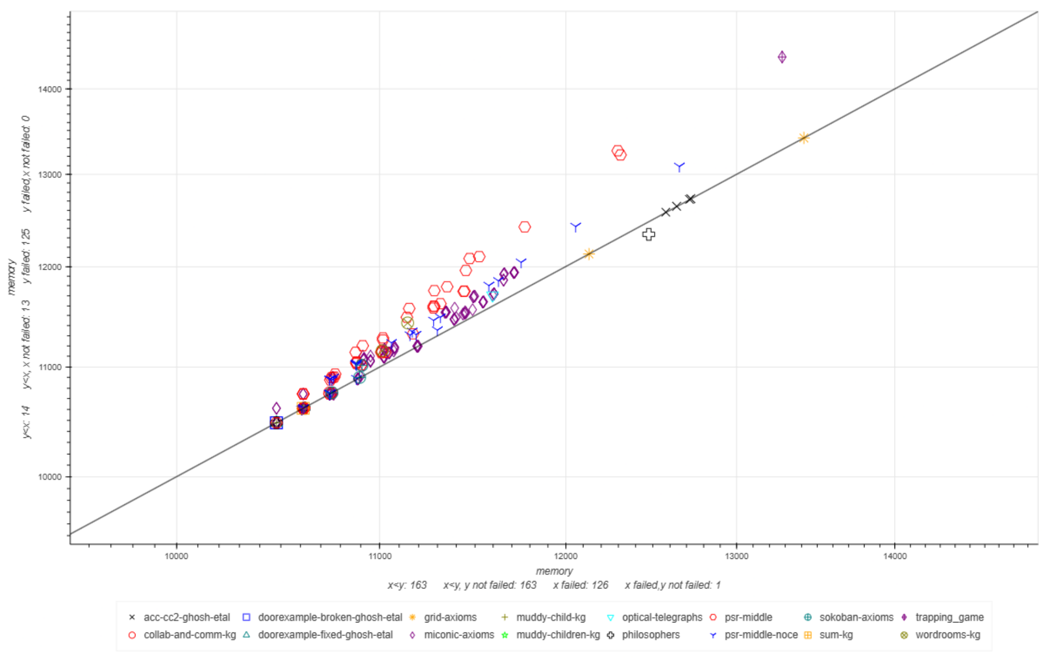


Figure 5.17: figure

add, negative

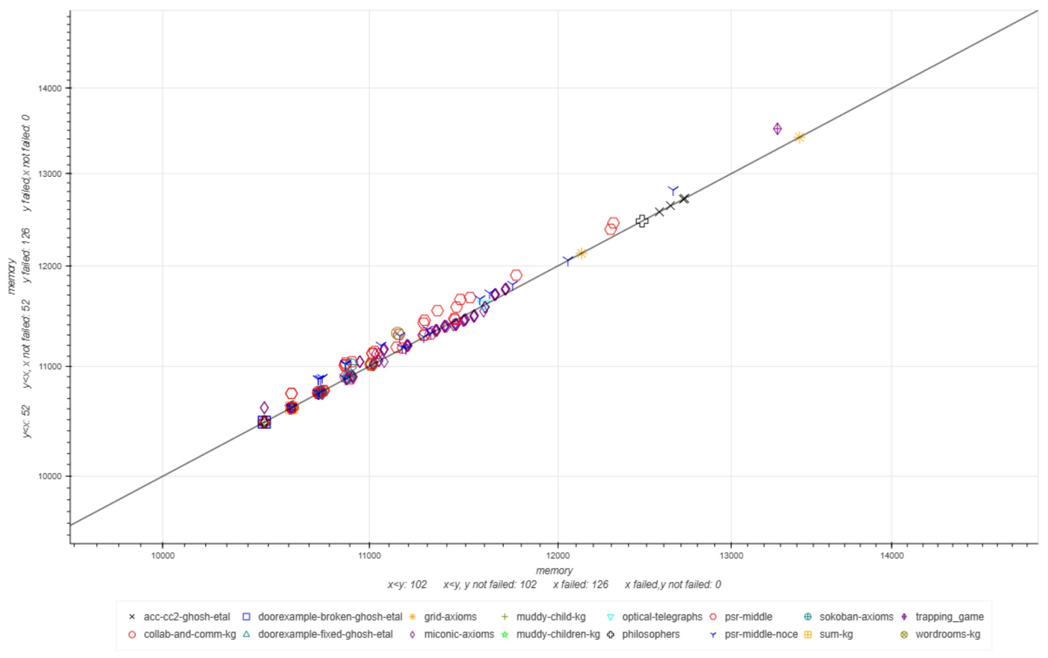


Figure 5.18: figure

add, negative, no search

results/m add tseitin negative cycles.png

Figure 5.19: figure

add, negative cycles

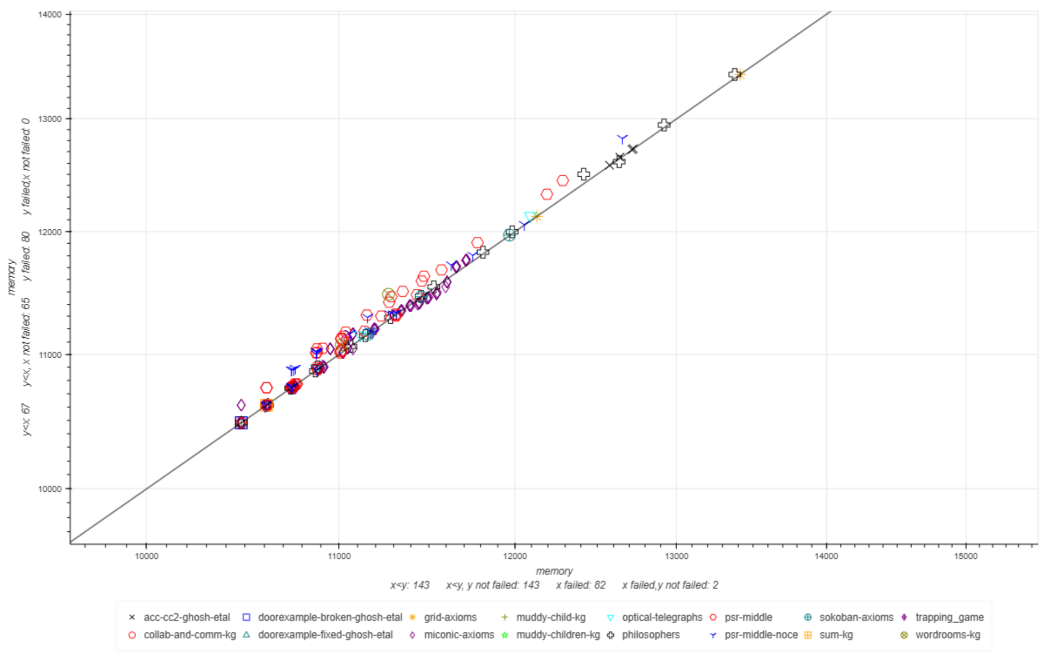


Figure 5.20: figure

add, negative cycles, no search

Similar to the runtime results discussed above, the transformation shows a slight negative impact on memory usage. On average, the measurements lie above the central line, indicating that our transformation generally leads to higher memory consumption.

6 Discussion and Conclusion

In comparison to the work of [2], we found that the transformation applied at the level of the search component does **not** increase the coverage of benchmark domains, by maintaining at least equal coverage overall, indicating that the transformation of FDR axioms is not the reason for Borgwardt et al. [2] observed impact of the Tseitin transformation.
On the other hand, our empirical analysis shows that the Tseitin-inspired transformation on FDR axioms does not lead to measurable performance improvements. On the contrary, the results suggest a slight negative effect on both runtime and memory consumption.
One likely reason for this is the inherent trade-off introduced by the transformation: while we reduced the width of the FDR axioms, we also created new axioms. On the one hand, this can reduce memory usage if there are many redundant *bodys* in the axioms, where our duplicate elimination could have had a measurable impact. However, this effect was either not significant enough or such redundancies were not frequent enough to have an overall positive impact. On the other hand, the newly created axioms increase the memory required to store them. A similar trade-off appears in runtime: although the transformation reduces the number of checks per axiom with redundant *bodys*, it increases the number of axioms per layer that need to be evaluated during the fixed-point iteration, and the transformation itself also incurs additional computational cost.
Since there is **no** influence of our Tseitin transformation on the benchmark domain coverage, and the computational cost of memory and search time are influenced negatively, this transformation seems not to be efficient on the level of search component.

Bibliography

[1] Blai Bonet and Hector Geffner. Planning as heuristic search. In *Artificial Intelligence*, volume 129, pages 5–33, 2001.

[2] Stefan Borgwardt, Duy Nhu, and Gabriele Röger. Automated planning with ontologies under coherence update semantics (extended version), 2025. URL https://arxiv.org/abs/2507.15120.

[3] Stefan Edelkamp and Jörg Hoffmann. Pddl2.2: The language for the classical part of the 4th international planning competition. In *Proceedings of the ICAPS-2004 Workshop on the Competition*, pages 1–14, 2004.

[4] Alfonso Gerevini, Alessandro Saetti, Ivan Serina, and Paolo Toninelli. Fast planning in domains with derived predicates: An approach based on rule-action graphs and local search. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, 2005.

[5] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[6] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[7] Malte Helmert. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173(5–6):503–535, 2009.

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

[9] Jörg Hoffmann, Stefan Edelkamp, Sylvie Thiébaux, Roman Englert, Frederico Liporace, and Sebastian Trüg. Engineering benchmarks for planning: the domains used in the deterministic part of ipc-4. *Journal of Artificial Intelligence Research*, 26:453–541, 2006.

[10] Franc Ivankovic and Patrik Haslum. Optimal planning with axioms. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[11] Franc Ivankovic, Patrik Haslum, and Takuya Miura. Axiom benchmarks for planning. https://github.com/dosydon/axiom_benchmarks, 2020. Accessed: 2025-08-02.

[12] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, 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, 1998. Report of the AIPS-98 Planning Competition Committee.

[13] Shuwa Miura. Planning benchmarks repository. https://dosydon.github.io/, 2023. Accessed: 2025-08-20.

[14] David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, 1986.

[15] Jussi Rintanen. Regression for classical and nondeterministic planning. *Journal of Artificial Intelligence Research*, 32:559–606, 2008.

[16] Jendrik Seipp, Florian Pommerening, Silvan Sievers, and Malte Helmert. Downward Lab. https://doi.org/10.5281/zenodo.790461, 2017.

[17] David Speck, Florian Geißer, Robert Mattmüller, and Álvaro Torralba. Symbolic planning with axioms. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 464–472, 2019.

[18] Sylvie Thiébaux, Jörg Hoffmann, and Bernhard Nebel. In defense of PDDL axioms. *Artificial Intelligence*, 168(1):38–69, 2005. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2005.05.004. URL https://www.sciencedirect.com/science/article/pii/S0004370205000810.

[19] G. S. Tseitin. On the complexity of derivation in propositional calculus. In A. O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic, Part II*, pages 115–125. Consultants Bureau, 1968. English translation; originally 1968 (Russian).

[20] David E Wilkins. Prism: Planning and scheduling for manufacturing. In *Proceedings of the Second International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 322–330, 1990.