OBSERVATIONAL EQUIVALENCE FOR MILP FORMULATIONS: BEHAVIORAL EVALUATION VIA SKETCH-AND-PROVE FORMAL VERIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixed-integer linear programming (MILP) formulations are widely used in scheduling, logistics, finance, and other domains. The same optimization problem can be represented by different formulations, which makes automatic correctness evaluation difficult. To evaluate formulation equivalence more appropriately, we propose a behavioral evaluation criterion for MILP formulations, which, building on the notion of observational equivalence, compares two formulations through their observable optimality outcomes under any problem parameter. We realize this criterion with a sketch-and-prove process that produces a proof certified by Lean. The process first constructs an informal plan and a formal proof sketch, and then automatically identifies the unproved goals and proves them one by one. Finally, we extend the benchmark for MILP formulation equivalence verification with nine classes of model transformations. Experiments show that our evaluation criterion overcomes key limitations of existing evaluation methods by assessing formulation correctness at the formulation level through the observable optimal outcomes. This makes the criterion suitable for the practical evaluation of both automatically optimization modelling and their subsequent transformations.

1 INTRODUCTION

Mixed-integer linear programming (MILP) is widely used in decision-making problems such as scheduling, power-system operations, and financial optimization (Ku & Beck, 2016; Knueven et al., 2020; Puerto et al., 2020). Recent advances in large language models (LLMs) have made it possible to formulate optimization problems directly from natural-language descriptions (Ramamonjison et al., 2022; Ahmaditeshnizi et al., 2024; Zhou et al., 2026; Chen et al., 2025). The same optimization problem, however, can admit many valid formulations, which differ in their decision variables, constraints, and model structures (Vielma, 2015; Lusby et al., 2020), and such formulations are routinely transformed to improve computational performance, for example by changing variables, lifting, adding cutting planes, or decomposing the problem (Liberti, 2009; Benders, 1962; Marchand et al., 2002; Balas et al., 1996). Consequently, one natural-language problem description can correspond to many formulations that are all valid, yet structurally different from one another.

This flexibility makes evaluation difficult: a transformation can change a formulation's representation without changing the optimization problem it describes. Existing evaluation methods often compare formulations through their syntax, structure, or outputs on sampled instances. Canonical methods match objective and constraint declarations (Ramamonjison et al., 2022), graph-based methods compare structural representations (Xing et al., 2024; Wang et al., 2025), and execution-based methods compare optimal objective values on instantiated problems (Ahmaditeshnizi et al., 2024). These methods can be sensitive to representation changes or provide only instance-specific evidence.

EquivaMap (Zhai et al., 2025) goes further by explicitly constructing mappings between the decision variables of two formulations and checking whether optimal solutions correspond under the mapping. However, this verification is still performed on individual problem instances and therefore does not establish equivalence over the full parameter space. FLARE (Robbins et al., 2026) moves from instance-level checking to formulation-level verification. It formalizes a constructive notion

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

of MILP reformulation in Lean and seeks explicit mappings between the parameter spaces, feasible solutions, and objective values of two formulations, allowing the resulting equivalence claim to hold over parameterized problem families. However, its criterion requires these mappings to hold over the entire feasible region. This requirement can be restrictive for valid reformulations that change the feasible representation, such as removing feasible points that can never be optimal, while preserving the observable optimal behavior. This leaves a gap for formulation-level evaluation that compares optimal outcomes over the entire parameter space. Such an evaluation is needed to determine whether two formulations preserve the same optimization behavior, even when their internal mathematical representations differ.

Observational equivalence is a standard notion in programming-language semantics, where two programs are considered equivalent when they cannot be distinguished through a specified set of observations (Morris, 1969). This view provides a natural way to compare different mathematical formulations of the same optimization problem. In parametric optimization, the optimal value function and the optimal solution mapping describe how the optimal outcome changes with problem parameters (Bonnans & Shapiro, 2000; Dua & Pistikopoulos, 2000). Motivated by this connection, we take the optimal solution set and the optimal objective value as the observable outcomes of a parameterized MILP formulation, and consider two formulations observationally equivalent when their observable outcomes can be mapped to each other. This criterion relaxes FLARE's constructive reformulation, which requires an explicit correspondence between the full feasible regions rather than only the observable outcomes.

We introduce observational equivalence into the evaluation of MILP formulations and develop a sketch-and-prove framework for certifying formulation equivalence. Our contributions are summarized as follows:

- We introduce **observational equivalence** for MILP formulations as a behavioral evaluation criterion. It evaluates formulation correctness at the formulation level through the observable optimization outcomes, namely the optimal solution set and the optimal objective value.
- We formalize observational equivalence and its necessary-and-sufficient certificate in Lean, and adapt the Draft, Sketch, and Prove paradigm (Jiang et al., 2023) to automatically construct proofs of MILP formulation equivalence. The procedure combines domain-specific proof planning with the decomposition of unresolved proofs into subproblems that are proved individually. The procedure raises the success rate of formally proving equivalence between MILP formulations.
- We construct **MILP-EquivalBench**, a benchmark for MILP formulation equivalence verification covering nine classes of model transformations. Experiments show that our behavioral evaluation criterion overcomes key limitations of existing evaluation methods. This supports practical correctness evaluation throughout automated optimization modeling and formulation transformations.

2 Related Work

2.1 MILP Formulations and Parametric Optimization

A mixed-integer linear program (MILP) can be written in the following standard form (Vielma, 2015):

$$\begin{aligned} \min_{x} \quad & c^\top x \\ \text{s.t.} \quad & Ax \le b, \quad \ell \le x \le u, \quad x \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_i} \end{aligned}$$

where x is the decision vector, $c \in \mathbb{R}^n$ is the cost coefficient vector, $A \in \mathbb{R}^{m \times n}$ is the constraint coefficient matrix, $b \in \mathbb{R}^m$ is the right-hand side vector, and $\ell, u \in \mathbb{R}^n$ with $\ell \le u$ are the lower and upper bounds on the variables. The total dimension $n = n_c + n_i$ is partitioned into n_c continuous variables ($x \in \mathbb{R}^{n_c}$) and n_i integer variables ($x \in \mathbb{Z}^{n_i}$). The objective $c^\top x$ and all constraints $Ax \le b$ are linear, which is the defining property of an MILP.

In the theory of parametric optimization, an MILP formulation is viewed as a family of optimization problems indexed by a parameter p belonging to a parameter space $\mathcal{P}$:

$$\mathcal{M}(p): \quad \min_{x} f(p, x) \quad \text{s.t.} \quad x \in \mathcal{F}(p).$$

Table 1: Correctness of representative MILP formulation equivalence evaluation methods under different model transformations. A checkmark indicates that the method correctly determines whether the transformed formulation is equivalent to the reference formulation.

Method	Objective scaling	Auxiliary variables	Valid cut (removing only globally non-optimal feasible points)	Invalid cut (removing points optimal under some parameters)
Canonical	×	×	×	✓
Graph-based	×	×	×	✓
Execution	×	✓	✓	×
EquivaMap	✓	✓	✓	×
FLARE	✓	✓	×	✓
Ours	✓	✓	✓	✓

where $\mathcal{F}(p) \subseteq \mathbb{R}^{n_c(p)} \times \mathbb{Z}^{n_i(p)}$ is a polyhedral feasible set whose coefficients are functions of p, and $f(p, \cdot)$ is a linear objective that also depends on p. Different formulations of the same optimization problem may introduce different decision variables and constraint structures, leading to different parameter spaces and feasible-set descriptions.

Given a parameterized MILP $\mathcal{M}$, its behaviour is characterized by the optimal value function

$$v_{\mathcal{M}}(p) := \inf\{\, f(p, x) \mid x \in \mathcal{F}(p) \,\}$$

and the optimal solution

$$S_{\mathcal{M}}(p) := \{\, x \in \mathcal{F}(p) \mid f(p, x) = v_{\mathcal{M}}(p) \,\}.$$

These objects are the central subjects of parametric optimization and multiparametric programming, which study how the optimal value and optimal solutions change as the parameters vary (Bonnans & Shapiro, 2000; Dua & Pistikopoulos, 2000).

2.2 MILP Formulation Evaluation and Equivalence

Several automated approaches have been developed to evaluate the correctness of generated or transformed MILP formulations. Canonical evaluation, as used in NL4OPT (Ramamonjison et al., 2022), compares a generated formulation with a reference by matching objective and constraint declarations, but it is sensitive to syntactic and structural differences between valid formulations. Graph-based methods (Xing et al., 2024; Wang et al., 2025) represent formulations as graphs and compare their structural representations, but transformations that introduce variables or constraints may change the graph even when the formulations remain semantically equivalent. Execution-based methods (Ahmaditeshnizi et al., 2024) compare optimal objective values on concrete instances, but agreement on sampled instances does not provide a formulation-level guarantee. EquivaMap (Zhai et al., 2025) further constructs symbolic mappings between variable spaces and verifies feasibility and optimality under those mappings, but the verification itself remains instance-specific. ModelEquivBench (Zhu & Xu, 2026) provides a richer profile of multiple certified semantic relations, although its guarantees depend on the supported mapping families and certificate schemas. FLARE (Robbins et al., 2026) provides formulation-level formal guarantees in Lean, but its constructive criterion requires mappings over the entire feasible region and can therefore reject valid transformations that remove non-optimal feasible points while preserving all optimal outcomes. Table 1 illustrates these limitations using several representative model transformations.

2.3 LLM-Assisted Formal Theorem Proving

Recent work has explored using large language models to automate formal theorem proving in Lean and other proof assistants. The DSP (Draft, Sketch, and Prove) framework (Jiang et al., 2023) first translates an informal proof into a formal proof sketch with gaps, then uses an automated solver to fill those gaps. Prover Agent (Baba et al., 2026) extends this idea by iteratively refining the proof based on Lean compiler feedback and generating auxiliary lemmas when a direct proof attempt fails. APOLLO (Ospanov et al., 2025) further introduces a staged repair pipeline that decomposes failed

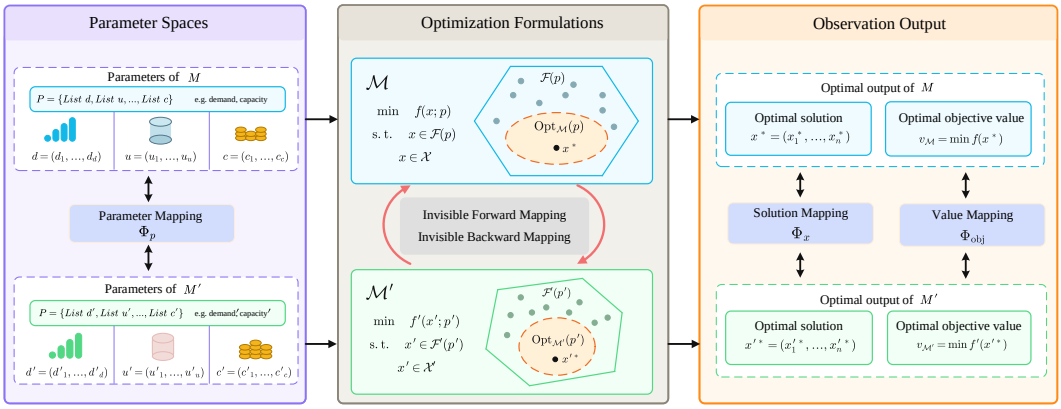


Figure 1: Illustration of observational equivalence for MILP formulations. Two formulations are observationally equivalent if their parameter spaces can be related by mappings such that, for every instance in either parameter space, the corresponding optimal outcomes can be mapped to each other. The observable outcomes consist of the optimal solution set and the optimal objective value.

proof attempts into smaller subproblems. Goedel-Architect (Chung et al., 2026) takes a planning-oriented approach by constructing a blueprint of definitions and lemmas before formal proof generation. These methods are primarily developed for formal proofs of mathematical theorems, but their proof-planning and decomposition strategies can also be adapted to domain-specific verification tasks. We apply these ideas to the formal verification of equivalence between MILP formulations, with the proof target defined by our observational equivalence certificate.

3 Method

This section presents our method for evaluating and formally verifying the equivalence of MILP formulations. We first introduce observational equivalence as a behavioral evaluation criterion and give its formal certificate in Section 3.1. We then describe sketch-and-prove procedure for automatically constructing Lean proofs of the certificate in Section 3.2.

3.1 The Observational Equivalence Criterion

3.1.1 MILP Formulations

We adopt the mathematical representation of a parameterized MILP formulation from FLARE (Robbins et al., 2026). A MILP formulation $\mathcal{M}$ is a tuple $(\mathcal{P}, X, f)$ where $\mathcal{P}$ is the parameter space, $X = \bigcup_{p \in \mathcal{P}} X_p$ indexes the decision variable spaces, and $f(p, x)$ is the linear objective function. The feasible set $\mathcal{F}_{\mathcal{M}}(p) \subseteq X_p$ is defined by linear constraints whose coefficients depend on p. The optimal value and the optimal solution set are denoted by $v_{\mathcal{M}}(p)$ and $S_{\mathcal{M}}(p)$ respectively, as defined in Section 2.1. We consider three types of mappings between two formulations $M = (P, X, f)$ and $M' = (P', X', f')$:

$$\Phi(M, M') = (\Phi_p, \Phi_x, \Phi_{\mathrm{obj}}),$$

corresponding to parameter mappings, solution mappings, and objective-value mappings, respectively. Each mapping is defined in both the forward and backward directions. For the parameter spaces, we use

$$\Phi_p^{\mathrm{fwd}} : P \to P', \qquad \Phi_p^{\mathrm{bwd}} : P' \to P.$$

For corresponding parameters $p \in P$ and $p' \in P'$, the solution mappings are

$$\Phi_x^{\mathrm{fwd}}(\cdot; p) : X_p \to X'_{p'}, \qquad \Phi_x^{\mathrm{bwd}}(\cdot; p') : X'_{p'} \to X_p.$$

For the objective values, we use

$$\Phi_{\mathrm{obj}}^{\mathrm{fwd}} : \mathbb{R} \to \mathbb{R}, \qquad \Phi_{\mathrm{obj}}^{\mathrm{bwd}} : \mathbb{R} \to \mathbb{R}.$$

Both objective-value mappings strictly preserve objective ordering.

3.1.2 OBSERVATIONAL EQUIVALENCE OF MILP FORMULATIONS

Observational equivalence is a standard notion in programming-language semantics: two programs are equivalent if no observation can distinguish between them (Morris, 1969). It compares programs by the behavior they expose rather than by how that behavior is computed, allowing different implementations to be regarded as equivalent.

Definition 1 (Observational Equivalence in Programming-Language Semantics)**.** Let $\mathcal{L}$ be a typed programming language equipped with an evaluation relation that maps a program to a value, together with a function obs that extracts the observable part of a value. Let e_1 and e_2 be two terms of the same type τ, with $\Gamma \vdash e_1 : \tau$ and $\Gamma \vdash e_2 : \tau$. They are observationally equivalent, written $e_1 \simeq_{\mathrm{ctx}} e_2 : \tau$, if for every context C, that is, every term with a single hole such that $C[e_1]$ and $C[e_2]$ are programs,

- $C[e_1]$ diverges if and only if $C[e_2]$ diverges;
- whenever both terminate, with values v_1 and v_2 respectively, $obs(v_1) = obs(v_2)$.

To adapt observational equivalence to evaluate MILP formulations, we regard a MILP formulation as a black box: its internal representation consists of its decision variables, constraints, and feasible set, while the behavior exposed under a parameter p is its optimal outcome, namely the optimal solution set $S_M(p)$ and the optimal objective value $v_M(p)$. This parallels observational equivalence in programming-language semantics, where programs may differ in their internal implementations but are compared through the behavior they expose in program contexts. Similarly, two MILP formulations may have different internal mathematical representations, while their equivalence is determined by the correspondence between their observable optimal outcomes. Figure 1 illustrates this definition for two parameterized MILP formulations, including the mappings between their parameter spaces, optimal solutions, and optimal objective values.

Definition 2 (Observational Equivalence of MILP Formulations)**.** Let $M = (P, X, f)$ and $M' = (P', X', f')$ be two MILP formulations. We say that M and M' are *observationally equivalent*, written $M \simeq M'$, if there exists $\Phi(M, M') = (\Phi_p, \Phi_x, \Phi_{\mathrm{obj}})$ satisfying:

- The optimal solutions of the two formulations can be mapped to each other:
$$\forall p \in P,\ \forall x \in S_M(p), \qquad \Phi_x^{\mathrm{fwd}}(x; p) \in S_{M'}\big(\Phi_p^{\mathrm{fwd}}(p)\big)\,,$$
$$\forall p' \in P',\ \forall x' \in S_{M'}(p'), \qquad \Phi_x^{\mathrm{bwd}}(x'; p') \in S_M\big(\Phi_p^{\mathrm{bwd}}(p')\big)\,.$$
- The optimal objective values of the two formulations can be mapped to each other:
$$\forall p \in P, \qquad v_{M'}\big(\Phi_p^{\mathrm{fwd}}(p)\big) = \Phi_{\mathrm{obj}}^{\mathrm{fwd}}(v_M(p))\,,$$
$$\forall p' \in P', \qquad v_M\big(\Phi_p^{\mathrm{bwd}}(p')\big) = \Phi_{\mathrm{obj}}^{\mathrm{bwd}}(v_{M'}(p'))\,.$$

Definition 2 directly quantifies over the optimal solution sets $S_M(p)$ and $S_{M'}(p')$. Proving it therefore requires reasoning about optimality for every parameter instance, which is difficult for automated proof synthesis because numerical solvers provide optimal solutions only for fixed instances. We instead characterize the same equivalence through conditions over feasible sets. The resulting certificate replaces direct optimality reasoning with feasibility preservation and objective inequalities, and can be machine-checked without solving either formulation.

Definition 3 (Observational Equivalence Certificate)**.** Let $M = (P, X, f)$ and $M' = (P', X', f')$ be two MILP formulations. We consider minimization formulations and assume that the optimum is attained for every parameter instance under consideration. An *observational equivalence certificate* between M and M' is a mapping $\Phi(M, M') = (\Phi_p, \Phi_x, \Phi_{\mathrm{obj}})$ satisfying:

- The feasible solutions of the two formulations can be mapped to each other:
$$\forall p \in P,\ \forall x \in \mathcal{F}_M(p), \qquad \Phi_x^{\mathrm{fwd}}(x; p) \in \mathcal{F}_{M'}\big(\Phi_p^{\mathrm{fwd}}(p)\big)\,,$$
$$\forall p' \in P',\ \forall x' \in \mathcal{F}_{M'}(p'), \qquad \Phi_x^{\mathrm{bwd}}(x'; p') \in \mathcal{F}_M\big(\Phi_p^{\mathrm{bwd}}(p')\big)\,.$$
- The objective values do not increase under the mappings in either direction:
$$\forall p \in P,\ \forall x \in \mathcal{F}_M(p), \qquad f'\big(\Phi_p^{\mathrm{fwd}}(p), \Phi_x^{\mathrm{fwd}}(x; p)\big) \le \Phi_{\mathrm{obj}}^{\mathrm{fwd}}(f(p, x))\,,$$
$$\forall p' \in P',\ \forall x' \in \mathcal{F}_{M'}(p'), \qquad f\big(\Phi_p^{\mathrm{bwd}}(p'), \Phi_x^{\mathrm{bwd}}(x'; p')\big) \le \Phi_{\mathrm{obj}}^{\mathrm{bwd}}(f'(p', x'))\,.$$

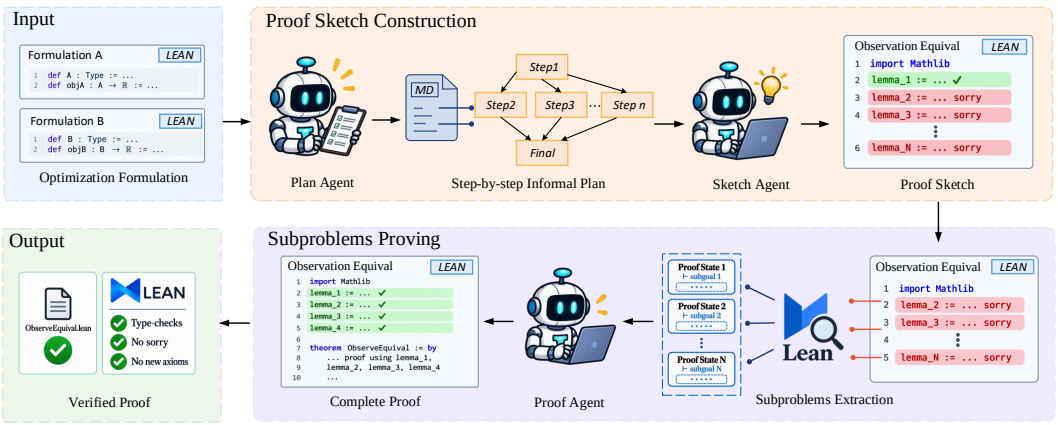


Figure 2: Overview of the sketch-and-prove framework. Given two MILP formulations formalized in Lean, the first stage constructs a step-by-step natural-language proof plan and converts it into a compilable Lean proof sketch with unresolved proof blocks. The second stage extracts these blocks as subproblems, proves them individually with a proof agent, and assembles the results into a complete Lean proof of observational equivalence.

The certificate conditions in Definition 3 are necessary and sufficient for the observational equivalence in Definition 2. By mapping feasible solutions in both directions while ensuring that their objective values do not worsen, the certificate guarantees that optimal solutions are mapped to optimal solutions and that the corresponding optimal objective values are preserved. The detailed proof is provided in Appendix B. Definition 3 is also the direct target of the machine-checkable proofs synthesized by our procedure. The mapping $\Phi(M, M')$ and its feasibility and objective conditions are formalized in Lean. A synthesized proof is accepted only when all proof obligations are discharged, with no remaining unproved goals or additional assumptions.

3.2 Sketch-and-Prove Proof Framework

Figure 2 provides an overview of the framework. Our framework follows the Draft, Sketch, and Prove (DSP) paradigm (Jiang et al., 2023) and adapts it to the verification of MILP observational equivalence. The procedure consists of two stages, *Sketch* and *Prove*. The Sketch stage is domain-specific: a Plan Agent first determines the mappings between the two MILP formulations and develops a step-by-step equivalence proof plan, after which a Sketch Agent constructs a compilable Lean certificate sketch. In the Prove stage, each unresolved proof obligation and its local proof state are extracted from Lean and assigned to an independent Proof Agent session.

3.2.1 Proof Sketch Construction

The first stage fixes the mathematical content of the proof before any proof is written. It proceeds in two steps: a natural-language plan, followed by a formal proof sketch built from that plan.

The planning agent reads the two MILP formulations and writes the informal plan. The plan identifies the mappings of Definition 3 and explains, how it can be proved step by step. The informal plan is described in natural language. Its role is to decide what must be shown and where the difficulty lies, and it leaves the formal part to the sketch.

Guided by the informal plan, the Sketch Agent constructs a Lean proof sketch specialized to the observational equivalence certificate. It instantiates the mappings identified in the plan and organizes the proof according to the feasibility and objective conditions in Definition 3. Conditions that can be discharged directly are proved in the sketch, while difficult intermediate facts are isolated as separate lemmas with `sorry` placeholders. Since each placeholder is attached to a concrete Lean proposition and local context, the resulting sketch provides a compilable certificate sketch whose unresolved propositions can be passed directly to the next stage.

3.2.2 SUBPROBLEM PROVING

The second stage solves the target propositions that the sketch leaves. Each target becomes a subproblem, and the proof agent solves each subproblem on its own.

For each `sorry` placeholder, the process queries the Lean language server for the goal at that position. The server returns the target proposition and the local context in which it must be proved. These two elements define the subproblem. The Proof Agent then works on this isolated goal by generating a Lean proof and checking it against the Lean language server. It iteratively revises the proof according to the returned proof state until the target is closed. When needed, the agent may introduce auxiliary lemmas, provided that all auxiliary results are also fully proved. Solving one typed proof obligation at a time restricts the formal search to a concrete target and its local context.

The proofs of all subproblems are assembled into the certificate by replacing the corresponding `sorry` placeholders. The resulting Lean file is compiled and type-checked, and we additionally verify that no `sorry` remains and no additional axioms are introduced. When both checks pass, the file is a machine-checkable certificate of the observational equivalence of the two formulations. Otherwise the two formulations are reported as not proved equivalent.

4 EXPERIMENTS

We conduct experiments to answer the following questions.

RQ1: Evaluation effectiveness. How accurately does our method distinguish equivalent and non-equivalent MILP formulations compared with existing evaluation methods?

RQ2: Framework components. How do the components of sketch-and-prove framework contribute to the verification performance?

RQ3: Model dependence. How sensitive is the verification performance to the choice of the underlying agent harness and language model?

4.1 EXPERIMENTAL SETUP

Dataset. We construct **MILP-EquivalBench**, a benchmark of 15 MILP problems. Each problem provides a reference formulation and 9 candidate formulations that introduce classes of transformations covering variable lifting, cutting-plane additions, modifications to the objective function and so on. This yields 135 formulation pairs (105 true, 30 false). Every formulation in this set is formalized in Lean, and every true pair is accompanied by a Lean certificate. Detailed problem scenarios and transformation categories are provided in Appendix D.

In addition, we evaluate on a subset of the FormulationBench (Robbins et al., 2026) that contains the 54 pairs (42 true, 12 false) previously studied by FLARE. These pairs come from 20 MILP problems covering scheduling, logistics, and facility-location domains, each with a reference formulation.

Baselines. For brevity, we refer to our observational equivalence criterion as **OBEquival** in the experiments. We compare OBEquival with four evaluation methods:

- **Execution** (Ahmaditeshnizi et al., 2024) solves the two formulations on the given problem instances and declares a pair equivalent when their optimal objective values agree.
- **ORGEval** (Wang et al., 2025) represents each formulation as a bipartite graph of variables and constraints, and declares a pair equivalent when the variable and constraint counts agree and the Weisfeiler–Lehman colourings of the two graphs match.
- **EquivaMap** (Zhai et al., 2025) prompts an LLM to propose a mapping from the variables of one formulation to linear expressions in the variables of the other, and verifies feasibility and optimality under this mapping on a given instance.
- **FLARE** (Robbins et al., 2026) requires explicit forward and backward mappings between the feasible sets together with a strictly increasing objective map, and accepts a pair when it produces a Lean proof of constructive reformulation.

Table 2: Comparison of evaluation methods on MILP formulation benchmarks. TP, FP, TN, and FN are summed over three runs, while Precision, Recall, and Accuracy are averaged over the three runs.

	FormulationBench							MILP-EquivalBench						
Method	TP	FP	TN	FN	Precision	Recall	Accuracy	TP	FP	TN	FN	Precision	Recall	Accuracy
Execution	117	21	15	9	84.8%	92.9%	81.5%	270	0	90	45	100%	85.7%	88.9%
ORGEval	12	0	36	114	100%	9.5%	29.6%	0	0	90	315	–	0%	22.2%
EquivaMap	120	15	21	6	88.9%	95.2%	87.0%	291	45	45	24	86.6%	92.4%	83.0%
FLARE	90	0	36	36	100%	71.4%	77.8%	270	0	90	45	100%	85.7%	88.9%
OBEquival	125	0	36	1	**100%**	**99.2%**	**99.4%**	315	0	90	0	**100%**	**100%**	**100%**

Metrics. Each method returns a decision for every pair, which we compare with the label of that pair. A true positive is a true pair that the method accepts, a false positive is a false pair that it accepts, and true and false negatives are defined accordingly. With N the number of pairs, we report

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \qquad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \qquad \text{Accuracy} = \frac{\text{TP} + \text{TN}}{N}.$$

The three metrics measure different aspects of a method's decisions. Precision is the fraction of the pairs that a method declares true which are in fact true. Recall is the fraction of the true pairs that it declares true. Accuracy is the fraction of all pairs whose label the method reproduces.

Implementation Details. Execution and EquivaMap use SCIP (Achterberg, 2009) for solver-based evaluation. EquivaMap uses GPT-5.6-SOL (OpenAI, 2026b) through API calls with medium reasoning effort to construct variable mappings. For FLARE and our method, the agent sessions run GPT-5.6-SOL in Codex (OpenAI, 2026a), also with medium reasoning effort. The prompts and agent instructions used in our proof procedure are provided in Appendix C.

4.2 Comparison with Existing Evaluation Methods

Table 2 summarizes the main results. Each method is run three times on each benchmark. We aggregate the sum of classification counts over the three runs and report the mean evaluation metrics. OBEquival achieves 100% Precision, 99.2% Recall, and 99.4% Accuracy on FormulationBench, and 100% on all three metrics on MILP-EquivalBench. In contrast, each baseline makes errors on at least one of the two benchmarks. We next analyze these errors according to the information and equivalence criteria used by each evaluation method.

Execution evaluation leads to two types of errors. First, equivalent formulations may have different objective values under valid objective transformations, causing them to be incorrectly rejected and lowering recall. Second, two non-equivalent formulations may happen to have the same optimal value on the evaluated instance, while differing under other parameter settings. Such pairs may be incorrectly accepted, leading to false positives and lower precision. **ORGEval** compares the structural representations of the two formulations. Equivalent transformations that introduce auxiliary variables or additional constraints change the corresponding graph structure, causing many equivalent pairs to be incorrectly rejected. **EquivaMap** constructs a variable mapping and verifies feasibility and optimality on individual instances. However, a mapping that succeeds on the evaluated instance does not guarantee equivalence under other parameter settings. As a result, non-equivalent pairs may be incorrectly accepted when their inconsistency appears only on other instances, leading to false positives and lower precision.

FLARE produces false negatives on both benchmarks, but for different reasons. On FormulationBench, the false negatives mainly arise from the formal proof-synthesis process: although the formulation pairs satisfy FLARE's constructive reformulation criterion, the agent fails to complete the corresponding Lean proof. On MILP-EquivalBench, the false negatives instead reflect a limitation of the criterion itself. FLARE requires forward and backward mappings over the entire feasible region, and can therefore reject valid reformulations that remove non-optimal feasible points while preserving all observable optimal outcomes.

Compared with the baselines, **OBEquival (ours)** avoids their main failure modes while retaining a proof for every accepted pair. It produces no false positives on either benchmark and achieves the

Table 3: Ablation study of different agent–model configurations for OBEquival.

		FormulationBench							MILP-EquivalBench						
Agent	Model	TP	FP	TN	FN	Precision	Recall	Accuracy	TP	FP	TN	FN	Precision	Recall	Accuracy
DSH	DeepSeek-V4.1-Flash	78	0	36	48	100%	61.9%	70.4%	273	0	90	42	100%	86.7%	89.6%
ZCode	GLM-5.3-Flash	56	4	32	70	93.3%	44.4%	54.3%	172	2	88	143	98.8%	54.6%	64.2%
Codex	GPT-5.6-SOL	125	0	36	1	**100%**	**99.2%**	**99.4%**	315	0	90	0	**100%**	**100%**	**100%**

Table 4: Ablation study of different components in OBEquival.

	FormulationBench							MILP-EquivalBench						
Method	TP	FP	TN	FN	Precision	Recall	Accuracy	TP	FP	TN	FN	Precision	Recall	Accuracy
Without Informal Plan	118	0	36	8	100%	93.7%	95.1%	315	0	90	0	100%	100%	100%
Single Session Proof	115	0	36	11	100%	91.3%	93.2%	314	0	90	1	100%	99.7%	99.8%
Ours	125	0	36	1	**100%**	**99.2%**	**99.4%**	315	0	90	0	**100%**	**100%**	**100%**

highest recall and accuracy. On MILP-EquivalBench, all pairs are correctly classified, while on FormulationBench only one false negative is observed across the three runs. This false negative arises during proof synthesis: after the formal proof sketch is generated, one of the resulting subproblems cannot be successfully proved, preventing completion of the corresponding Lean certificate. Every pair accepted as equivalent is supported by a Lean certificate.

4.3 Ablation Study

We ablate the harness and the model that the framework uses, replacing both by the GLM harness ZCode (Z.AI, 2026) with GLM-5.3-Flash (GLM-5-Team et al., 2026), and by the DeepSeek harness dsh (DeepSeek-AI, 2026b) with DeepSeek-V4.1-Flash (DeepSeek-AI, 2026a). Table 3 shows that the verification performance varies across different agent and model configurations. Compared with the default Codex with GPT-5.6-SOL configuration, both the DeepSeek and GLM configurations perform worse on formal proof synthesis and more frequently fail to complete valid Lean proofs for equivalent formulation pairs. These proof-synthesis failures produce more false negatives, leading to lower recall and accuracy. In addition, the GLM configuration produces a small number of false positives, indicating that it is less reliable in adhering to the intended observational equivalence proof specification. These proof-synthesis failures and specification violations lead to lower overall verification performance.

We ablate the two components of the framework. The first component is the informal plan (*Without Informal Plan*). The second is the formal proof sketch together with the decomposition of the proof into subproblems. Removing this component leaves a single agent session that attempts the complete certificate (*Single Session Proof*). Table 4 shows that removing either component degrades verification performance on FormulationBench, with a larger drop for the *Single Session Proof* variant. This indicates that both the informal proof plan and the proof sketch with subproblem decomposition contribute to successful formal verification, while the latter has a stronger effect on difficult proofs. In contrast, both ablations have only a minor effect on MILP-EquivalBench. This is consistent with the design of MILP-EquivalBench. We focuses primarily on evaluating equivalence across representative formulation transformations rather than on challenging formal proof synthesis.

5 Conclusion

This work studies MILP formulation equivalence through observable optimal behavior. We use optimal solutions and objective values to define what should be preserved across different formulations, and formalize the resulting criterion in Lean. A sketch-and-prove procedure constructs machine-checkable equivalence certificates. We also introduce MILP-EquivalBench to cover diverse formulation transformations. Experiments on both benchmarks show that our criterion captures valid reformulations missed by existing methods while maintaining formal verification for accepted pairs. This supports more reliable evaluation of automatical optimization modelling and provides verification feedback for improving subsequent modeling and transformation.

AI USE STATEMENT

Generative AI tools were used as part of the proof-verification procedure evaluated in this work. Specifically, GPT-5.6-SOL, accessed through API calls and Codex, was used to generate informal proof plans, construct Lean proof sketches, and propose proofs for Lean proof. Generative AI tools were also used during benchmark construction to generate Lean representations of the MILP formulations. These formalizations were subsequently checked for compilation and used as the formal inputs to the verification experiments. Generative AI tools were additionally used during the research and writing process to assist with methodological discussion, mathematical formalization and proof development, experimental analysis, and manuscript drafting and editing. All AI-assisted text, mathematical claims, citations, experimental results, and proof artifacts were reviewed by the authors. The authors take responsibility for the final content of the paper and for all research artifacts produced with the aid of generative AI.

REFERENCES

Tobias Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009. doi: 10.1007/s12532-008-0001-1.

Ali Ahmaditeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 577–596. PMLR, 2024. URL https://proceedings.mlr.press/v235/ahmaditeshnizi24a.html.

Kaito Baba, Chaoran Liu, Shuhei Kurita, and Akiyoshi Sannai. Prover Agent: An agent-based framework for formal mathematical proofs, 2026. URL https://arxiv.org/abs/2506.19923.

Egon Balas, Sebastián Ceria, and Gérard Cornuéjols. Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42(9):1229–1246, 1996. doi: 10.1287/mnsc.42.9.1229.

J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962. doi: 10.1007/BF01386316.

J. Frédéric Bonnans and Alexander Shapiro. *Perturbation Analysis of Optimization Problems*. Springer Series in Operations Research. Springer, New York, 2000. doi: 10.1007/978-1-4612-1394-9.

Yitian Chen, Jingfan Xia, Siyu Shao, Dongdong Ge, and Yinyu Ye. Solver-informed rl: Grounding large language models for authentic optimization modeling. In *Advances in Neural Information Processing Systems*, volume 38, pp. 117446–117488, 2025. doi: 10.52202/085713-3539.

Yitian Chen, Cheng Cheng, Yinan Sun, Zi Ling, and Dongdong Ge. OPT-Engine: Benchmarking the limits of LLMs in optimization modeling via complexity scaling, 2026. URL https://arxiv.org/abs/2601.19924.

Jui-Hui Chung, Ziyang Cai, Zihao Li, Qishuo Yin, Rohit Agarwal, Simon Park, Rodrigo Porto, Narutatsu Ri, Ziran Yang, Shange Tang, Xingyu Dang, Hongzhou Lin, Mengdi Wang, Danqi Chen, Chi Jin, Liam H. Fowl, and Sanjeev Arora. Goedel-Architect: Streamlining formal theorem proving with blueprint generation and refinement, 2026. URL https://arxiv.org/abs/2606.06468.

DeepSeek-AI. Introducing deepseek-v4.1-flash: Smarter, faster, more efficient. https://www.deepseek.com/en/news/deepseek-v4-1-flash/, 2026a.

DeepSeek-AI. Deepseek harness: Everything is a plugin. https://github.com/deepseek-ai/deepseek-harness, 2026b.

Vivek Dua and Efstratios N. Pistikopoulos. An algorithm for the solution of multiparametric mixed integer linear programming problems. *Annals of Operations Research*, 99:123–139, 2000. doi: 10.1023/A:1019241000636.

Alan Frieze and Mark Jerrum. Improved approximation algorithms for MAX k-CUT and MAX BISECTION. *Algorithmica*, 18(1):67–81, 1997. doi: 10.1007/BF02523688.

GLM-5-Team, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, et al. GLM-5: From vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. In Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon (eds.), *Column Generation*, pp. 33–65. Springer, 2005. doi: 10.1007/0-387-25486-2_2.

Albert Q. Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs, 2023. URL https://arxiv.org/abs/2210.12283.

Bernard Knueven, James Ostrowski, and Jean-Paul Watson. On mixed-integer programming formulations for the unit commitment problem. *INFORMS Journal on Computing*, 32(4):857–876, 2020. doi: 10.1287/ijoc.2019.0944.

Wen-Yang Ku and J. Christopher Beck. Mixed integer programming models for job shop scheduling: A computational analysis. *Computers & Operations Research*, 73:165–173, 2016. doi: 10.1016/j.cor.2016.04.006.

Leo Liberti. Reformulations in mathematical programming: Definitions and systematics. *RAIRO - Operations Research*, 43(1):55–85, 2009. doi: 10.1051/ro/2009005.

Richard Martin Lusby, Mette Gamst, Stefan Røpke, and Simon Spoorendonk. Simultaneously exploiting two formulations: An exact benders decomposition approach. *Computers & Operations Research*, 123:105041, 2020. doi: 10.1016/j.cor.2020.105041.

Hugues Marchand, Alexander Martin, Robert Weismantel, and Laurence Wolsey. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, 123(1-3):397–446, 2002. doi: 10.1016/S0166-218X(01)00348-1.

James Hiram Morris. *Lambda-Calculus Models of Programming Languages*. PhD thesis, Massachusetts Institute of Technology, 1969. URL http://hdl.handle.net/1721.1/64850.

G. L. Nemhauser and L. E. Trotter. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8(1):232–248, 1975. doi: 10.1007/BF01580444.

OpenAI. Codex, 2026a. OpenAI.

OpenAI. GPT-5.6: Frontier intelligence that scales with your ambition, 2026b. OpenAI.

Azim Ospanov, Farzan Farnia, and Roozbeh Yousefzadeh. APOLLO: Automated LLM and Lean collaboration for advanced formal reasoning, 2025. URL https://arxiv.org/abs/2505.05758.

Justo Puerto, Moisés Rodríguez-Madrena, and Andrea Scozzari. Clustering and portfolio selection problems: A unified framework. *Computers & Operations Research*, 117:104891, 2020. doi: 10.1016/j.cor.2020.104891.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pp. 189–203. PMLR, 2022. URL https://proceedings.mlr.press/v220/ramamonjison23a.html.

Henry Robbins, Connor Lawless, Madeleine Udell, and Ellen Vitercik. FLARE: Verifying MILP reformulations with LLM-based theorem proving, 2026. URL https://arxiv.org/abs/2608.25220.

Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1–2):59–66, 1956. doi: 10.1002/nav.3800030106.

Vijay V. Vazirani. *Approximation Algorithms*. Springer, Berlin, Heidelberg, 2001. ISBN 978-3-540-65367-7. doi: 10.1007/978-3-662-04565-7.

Juan Pablo Vielma. Mixed integer linear programming formulation techniques. *SIAM Review*, 57 (1):3–57, 2015. doi: 10.1137/130915303.

Zhuohan Wang, Ziwei Zhu, Ziniu Li, Congliang Chen, Yizhou Han, Yufeng Lin, Zhihang Lin, Angyang Gu, Xinglin Hu, Ruoyu Sun, and Tian Ding. ORGEval: Graph-theoretic evaluation of LLMs in optimization modeling, 2025. URL https://arxiv.org/abs/2510.27610.

Linzi Xing, Xinglu Wang, Yuxi Feng, Zhenan Fan, Jing Xiong, Zhijiang Guo, Xiaojin Fu, Rindra Ramamonjison, Mahdi Mostajabdaveh, Xiongwei Han, Zirui Zhou, and Yong Zhang. Towards human-aligned evaluation for linear programming word problems. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 16550–16556. ELRA and ICCL, 2024. URL https://aclanthology.org/2024.lrec-main.1438/.

Z.AI. Zcode: Official harness for glm-5.3. https://zcode.z.ai/en, 2026.

Haotian Zhai, Connor Lawless, Ellen Vitercik, and Liu Leqi. EquivaMap: Leveraging LLMs for automatic equivalence checking of optimization formulations. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 74288–74305. PMLR, 2025. URL https://proceedings.mlr.press/v267/zhai25a.html.

Chenyu Zhou, Tianyi Xu, Jianghao Lin, and Dongdong Ge. StepORLM: A self-evolving framework with generative process supervision for operations research language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ZrgxU8WMmG.

Penglin Zhu and Jungang Xu. ModelEquivBench: Certifying multi-relational evaluation of LLM-generated optimization models, 2026. URL https://arxiv.org/abs/2607.29431.

A Overview of Appendix

We organize the appendix into four parts. Appendix B gives the necessary and sufficient conditions for observation equivalence, with complete proofs, together with the Lean representation of the equivalence definitions. Appendix C documents the agent skills that drive the proof pipeline. Appendix D describes how the benchmark was constructed, namely the sources of the 15 problem types and the details of the nine transformations. Finally, Appendix E presents case studies that illustrate the limitations of each baseline method.

B Proofs and Formalization

B.1 Necessary and Sufficient Conditions for Observational Equivalence

This subsection proves the certificate equivalence pointwise for matched parameter pairs. The bidirectional statement in Definition 2 is obtained by applying the same directional argument in both directions, with the second direction obtained by exchanging M and M'.

Notation. Let $M = (P, X, f)$ and $M' = (P', X', f')$ be two MILP formulations, and let

$$\Phi(M, M') = (\Phi_p, \Phi_{\mathrm{fwd}}, \Phi_{\mathrm{bwd}}, \Phi_{\mathrm{obj}})$$

be a correspondence as defined in Section 3.1.2. For any $p \in P$, let $p' = \Phi_p(p)$.

We use $\mathcal{F}_M(p)$ and $\mathcal{F}_{M'}(p')$ to denote the feasible sets, $S_M(p)$ and $S_{M'}(p')$ to denote the optimal solution sets, and $v_M(p)$ and $v_{M'}(p')$ to denote the corresponding optimal values. Throughout this subsection, we assume that the optimum is attained for every parameter instance under consideration. For the directional argument below, fix a matched pair (p, p') satisfying $p' = \Phi_p^{\mathrm{fwd}}(p)$ and $\Phi_p^{\mathrm{bwd}}(p') = p$. We do not assume that the parameter maps are globally bijective. On the objective values realized by this matched pair, the two directional objective mappings are understood as descriptions of a common strictly increasing objective scale; equivalently, they agree with mutually consistent forward and backward maps on this realized range. No invertibility condition is imposed outside the matched parameter and objective-value ranges. The reverse direction is obtained by exchanging M and M' and swapping the forward and backward maps.

Sufficiency. Suppose there exists an observational equivalence certificate $\Phi(M, M')$ satisfying Definition 3. Fix any $p \in P$ and let $p' = \Phi_p(p)$.

First, take any $x^* \in S_M(p)$. By forward feasibility, $\Phi_{\mathrm{fwd}}(x^*; p) \in \mathcal{F}_{M'}(p')$. For any $y' \in \mathcal{F}_{M'}(p')$, backward feasibility gives $\Phi_{\mathrm{bwd}}(y'; p) \in \mathcal{F}_M(p)$. Since x^* is optimal,

$$f(x^*; p) \le f(\Phi_{\mathrm{bwd}}(y'; p); p).$$

Because Φ_{obj} preserves objective ordering, together with the two objective bounds we obtain

$$f'(\Phi_{\mathrm{fwd}}(x^*; p); p') \le \Phi_{\mathrm{obj}}(f(x^*; p)) \le \Phi_{\mathrm{obj}}\big(f(\Phi_{\mathrm{bwd}}(y'; p); p)\big) \le f'(y'; p').$$

Thus $\Phi_{\mathrm{fwd}}(x^*; p) \in S_{M'}(p')$.

Conversely, take any $y^* \in S_{M'}(p')$. By backward feasibility, $\Phi_{\mathrm{bwd}}(y^*; p) \in \mathcal{F}_M(p)$. For any $x \in \mathcal{F}_M(p)$, forward feasibility gives $\Phi_{\mathrm{fwd}}(x; p) \in \mathcal{F}_{M'}(p')$. Since y^* is optimal,

$$\Phi_{\mathrm{obj}}\big(f(\Phi_{\mathrm{bwd}}(y^*; p); p)\big) \le f'(y^*; p') \le f'(\Phi_{\mathrm{fwd}}(x; p); p') \le \Phi_{\mathrm{obj}}(f(x; p)).$$

As Φ_{obj} also reflects objective ordering, this implies

$$f(\Phi_{\mathrm{bwd}}(y^*; p); p) \le f(x; p),$$

and hence $\Phi_{\mathrm{bwd}}(y^*; p) \in S_M(p)$.

Finally, applying the objective bounds to $x^* \in S_M(p)$ and $y^* \in S_{M'}(p')$, and using the optimality established above, gives

$$v_{M'}(p') \le \Phi_{\mathrm{obj}}(v_M(p)) \le v_{M'}(p').$$

Therefore,

$$v_{M'}(p') = \Phi_{\mathrm{obj}}(v_M(p)).$$

Hence all conditions of Definition 2 hold, so $M \simeq M'$.

Necessity. Suppose that $M \simeq M'$ according to Definition 2, witnessed by an observation correspondence

$$\Phi(M, M') = (\Phi_p, \Phi_{\text{fwd}}, \Phi_{\text{bwd}}, \Phi_{\text{obj}}).$$

We construct an observational equivalence certificate with the same parameter mapping Φ_p and objective mapping Φ_{obj}, but possibly different forward and backward mappings.

For each $p \in P$, let $p' = \Phi_p(p)$. Since the optimum is attained in both formulations, choose

$$x_p^* \in S_M(p), \qquad y_p^* \in S_{M'}(p').$$

Define new mappings by

$$\widetilde{\Phi}_{\text{fwd}}(x; p) := y_p^* \quad \text{for all } x \in X_p, \qquad \widetilde{\Phi}_{\text{bwd}}(y; p) := x_p^* \quad \text{for all } y \in X'_{p'}.$$

Since x_p^* and y_p^* are optimal, they are feasible. Therefore, for any $x \in \mathcal{F}_M(p)$ and $y \in \mathcal{F}_{M'}(p')$,

$$\widetilde{\Phi}_{\text{fwd}}(x; p) = y_p^* \in \mathcal{F}_{M'}(p'), \qquad \widetilde{\Phi}_{\text{bwd}}(y; p) = x_p^* \in \mathcal{F}_M(p),$$

so the feasibility conditions of Definition 3 hold.

It remains to verify the objective bounds. By observational equivalence,

$$v_{M'}(p') = \Phi_{\text{obj}}(v_M(p)).$$

For any $x \in \mathcal{F}_M(p)$, optimality of x_p^* gives $v_M(p) \le f(x; p)$. Since Φ_{obj} preserves objective ordering,

$$\begin{aligned} f'(\widetilde{\Phi}_{\text{fwd}}(x; p); p') &= f'(y_p^*; p') \\ &= v_{M'}(p') \\ &= \Phi_{\text{obj}}(v_M(p)) \\ &\le \Phi_{\text{obj}}(f(x; p)). \end{aligned}$$

Hence the forward objective bound holds.

Similarly, for any $y \in \mathcal{F}_{M'}(p')$, $v_{M'}(p') \le f'(y; p')$. Since x_p^* is optimal for M at p,

$$\begin{aligned} \Phi_{\text{obj}}\big(f(\widetilde{\Phi}_{\text{bwd}}(y; p); p)\big) &= \Phi_{\text{obj}}(f(x_p^*; p)) \\ &= \Phi_{\text{obj}}(v_M(p)) \\ &= v_{M'}(p') \\ &\le f'(y; p'). \end{aligned}$$

Thus the backward objective bound also holds.

Therefore,

$$\widetilde{\Phi}(M, M') = (\Phi_p, \widetilde{\Phi}_{\text{fwd}}, \widetilde{\Phi}_{\text{bwd}}, \Phi_{\text{obj}})$$

satisfies Definition 3. Hence observational equivalence implies the existence of an observational equivalence certificate.

B.2 Lean Representation of the Equivalence Definitions

The MILP formulation and observational equivalence certificate are represented in Lean by the following two structures.

MILPFormulation

```
structure MILPFormulation where
  Params : Type
  Vars : Params → Type
  feasible : (p : Params) → Vars p → Prop
  obj : (p : Params) → Vars p → ℝ
```

MILPObservationEquivalence

```
structure MILPObservationEquivalence
    (F G : MILPFormulation) where
  paramMap : F.Params → G.Params
  fwd : (p : F.Params) →
    F.Vars p → G.Vars (paramMap p)
  bwd : (p : F.Params) →
    G.Vars (paramMap p) → F.Vars p

  fwd_feasible : ∀ p x, F.feasible p x →
    G.feasible (paramMap p) (fwd p x)

  bwd_feasible : ∀ p y, G.feasible (paramMap p) y →
    F.feasible p (bwd p y)

  objMap : ℝ → ℝ
  objMap_mono : StrictMono objMap

  fwd_obj_le : ∀ p x, F.feasible p x →
    G.obj (paramMap p) (fwd p x) ≤
      objMap (F.obj p x)

  bwd_obj_le : ∀ p y, G.feasible (paramMap p) y →
    objMap (F.obj p (bwd p y)) ≤
      G.obj (paramMap p) y
```

C AGENT SKILLS

We list the skills and prompts that the agent in our framework uses. They are the prompt that asks for a natural-language proof plan, the skill that lays out a compiling proof sketch with the remaining obligations isolated behind individual `sorry`s, the skill that closes those Lean goals one at a time, and the base skill that fixes the file and definition framework the certificate must follow.

C.1 INFORMAL PLAN

The informal plan prompt asks the agent to produce a natural-language proof plan establishing the observation equivalence, and to name the intermediate proof steps and lemmas it will need.

Informal plan skill

You are a mathematician. The current working directory holds two MILP formulations that have already been formalised in Lean 4 and frozen:

– ‘A/Formulation.lean‘ and ‘B/Formulation.lean‘ read these first;
– ‘Common.lean‘ defines ‘MILPFormulation‘ and the certificate structure ‘MILPObservationEquivalence‘;
– ‘A/formulation.md‘ and ‘B/formulation.md‘ informal descriptions;
– ‘map.md‘ / ‘map.py‘ the given parameter map, when present.

TASK. Write an informal, step−by−step mathematical proof that formulation B is observation equivalent to formulation A, and save it as ‘plan.md‘ in the current working directory.

Treat the four certificate obligations separately, one section each:

1. forward feasibility every A−feasible point maps to a B−feasible point;
2. backward feasibility every B−feasible point comes from an A−feasible point;
3. forward objective how the B objective at the image compares with the A objective;
4. backward objective the same comparison in the reverse direction.

State concretely the parameter map, the forward and backward variable maps, and the objective map (a single real function of the A objective value) that you would use. Then give the mathematical argument for each obligation, naming the intermediate claims (lemmas) the argument needs and saying why each one holds.

RULES.

– Write mathematics in English prose. Do NOT write Lean code, Lean syntax, or tactic names; the reader is a human, and a separate agent will do the formal work.
– Do NOT run ‘lake‘, ‘lean‘, ‘git‘, or any build, update or download command, and do not invoke the Lean server. Read the files you need directly. The working directory is a shared, pre−built project: running the toolchain here clones a dependency tree into it, wastes minutes, and can break the run.
– Use only assumptions that are actually present in the frozen Lean files.
– Be concrete and honest. If an obligation looks false, or you cannot find an argument for it, say so and explain precisely.

C.2 Formal Proof Sketch

The sketch prompt asks the agent to produce a proof sketch of the Lean formalization of equivalence. The generated Lean file must compile, each missing claim is left as a single `sorry`, and the intermediate claims are stated as separate lemmas.

Formal proof sketch skill

You are a mixed−integer linear programming (MILP) and Lean 4 expert. Two MILP formulations have ∗∗already been formalized∗∗ in Lean 4 by a trusted source. Your task now is to lay out the certificate’s ∗∗sketch∗∗ the maps and all four obligations and to mark the parts you cannot close yet with a single ‘sorry‘ each, so that a later step can attack those parts one at a time.
...
Frozen Formulations
– ‘A/Formulation.lean‘ and ‘B/Formulation.lean‘ are pre−built, trusted inputs and the ∗∗mathematical ground truth∗∗ of this task. The markdown files are informal commentary only: if a markdown sentence seems to disagree with the Lean, the Lean wins.
– Both files are ∗∗hash−verified after you finish∗∗. Any modification edit, overwrite, truncation, deletion, renaming voids the entire run. Do not touch them, and do not run commands that would rewrite them.
– Everything you need is already in the frozen files. You must not add assumptions, variables, constraints, or objective terms to either model. A mathematical fact that is missing is a ∗∗proof obligation∗∗: derive and prove it as a lemma inside ‘Reformulation.lean‘.
– Do not report ‘FORMALIZATION_GAP‘: the formalization phase is not part of this task and cannot be changed.
Parameter Map
The parameter mapping is given, not something you search for. Read ‘map.md‘ first and consult ‘map.py‘ where the notation is ambiguous. The ‘paramMap‘ field of the certificate must implement exactly this mapping between the frozen ‘Params‘ structures. The forward and backward variable maps are yours to find. Do not edit ‘map.md‘ or ‘map.py‘.
Informal plan available
‘plan.md‘ in the working directory is an informal (natural−language) proof plan prepared beforehand by a separate step. It states, for each of the four obligations, the maps to use and the mathematical argument, including the intermediate claims the argument needs. It carries no formal authority only Lean decides but it is a good source for the claims you should state as lemmas.
Your Task: the Sketch
Write ‘Reformulation.lean‘ so that it
1. declares the parameter map, the forward variable map, the backward variable map, the objective map (‘objMap : ‘, a single global function, no instance argument) together with ‘objMap_mono : StrictMono objMap‘;
2. provides all four obligations: ‘fwd_feasible‘, ‘bwd_feasible‘, ‘fwd_obj_le‘, ‘bwd_obj_le‘;
3. ∗∗compiles∗∗: ‘lake build‘ must succeed. ‘sorry‘ produces a warning, never an error, so a file with ‘sorry‘s must still compile. If you leave the file in a state that does not compile, the whole attempt is discarded;
4. closes whatever you can close right now, and leaves ∗∗exactly one ‘sorry‘∗∗ at each spot you cannot close.
The endpoint type must be exactly:
‘‘‘lean
MILPObservationEquivalence A.formulation B.formulation
‘‘‘
If the Four Conditions Cannot Hold
If you find a concrete counterexample, or a provable obstruction showing that the four conditions cannot hold for the frozen models, do ∗∗not∗∗ invent a proof and do not weaken the statement. Write
‘‘‘
NOT REFORMULATION: <the mathematical reason, in a sentence or two>
‘‘‘
as the ∗∗first line∗∗ of ‘Reformulation.lean‘ and stop. That is a legitimate outcome of this task: an honest refusal is worth more than a certificate that proves a weakened or vacuous statement.
Making the Holes Fine−Grained
The point of the sketch is that each ‘sorry‘ isolates one claim, so a later step can attack it in isolation with the goal state Lean reports at that position. Therefore:
– Do ∗∗not∗∗ leave one huge ‘sorry‘ for a whole obligation if you can say more. Write out the proof structure (the maps, the case split, the reduction) and put the ‘sorry‘ where a specific claim is missing.
– When an obligation needs an intermediate claim you cannot prove yet a counting argument, an ordering argument, a bound over all indices state that claim as its ∗∗own lemma∗∗ with its own hypotheses and conclusion, and leave the ‘sorry‘ inside that lemma’s proof. The isolated statement is far easier to finish than the surrounding goal.
– Prefer several small, well−named lemmas over one monolithic proof with one hole, but do not invent lemmas that are not needed by the argument.
Rules
– IMPORTANT: Only read/write files that exist in ∗this∗ working directory. Do not navigate outside of it for any reason.
– The Lean project root is the current directory. Use ‘import A.Formulation‘ and ‘import B.Formulation‘.
– You MUST use the lean−lsp MCP tools (mcp__lean−lsp__∗) to check compilation as you work; do not fall back to ‘lake env lean‘ for checking. ‘Bash(lake build:∗)‘ is permitted and safe.
– The certificate must be ∗∗honest∗∗: do not add hypotheses that A does not give you, do not replace a condition with a trivially true one, do not introduce ‘axiom‘s or ‘sorryAx‘−like escape hatches, and do not weaken ‘StrictMono‘ to ‘Monotone‘. A certificate that proves a weakened statement is worthless, and the axiom check catches custom axioms.

...

C.3 SUBPROBLEM PROVING

The goal state that Lean reports at the position of that `sorry` is given to the agent, and the agent is asked to close that one `sorry` only. The prompt allows the agent to add helper lemmas and any imports it needs, but forbids any modification to the rest of the file.

Subproblem proving skill

You are a mixed−integer linear programming (MILP) and Lean 4 expert. A Lean 4 certificate of observation equivalence between two MILP formulations is being built in this working directory. Its sketch compiles, and it has one open goal assigned to you.

The Open Goal

‘{{ rel }}‘ has an unfinished proof: the ‘sorry‘ at **line {{ hole_line }}, column {{ hole_column }}**. Lean reports this goal state at that position:

‘‘‘text
{{ goal_state }}
‘‘‘

In words, what has to be proved is:

‘‘‘text
{{ target }}
‘‘‘

Close **that goal**: replace that ‘sorry‘ with a real proof, and leave the file compiling.

What You May Change

− The ‘sorry‘ at line {{ hole_line }}, column {{ hole_column }}. This is your task.
− Helper declarations or ‘have‘ statements that you need, next to the goal or at the top level of the same namespace. Each must be fully proved (no ‘sorry‘). A helper need not be a sub−goal of the original argument: a useful special case, a monotonicity or exchange fact, a counting or ordering lemma, or an intermediate bound all count.
− Imports, if a Mathlib theory you need is not imported yet.

What You Must Not Change

− **Any other ‘sorry‘ in the file.** Other work is in flight on those. A goal closed by breaking another one is discarded.
− Any declaration that is already proved, including the rest of the certificate.
− ‘A/Formulation.lean‘, ‘B/Formulation.lean‘, ‘Common.lean‘, ‘lakefile.toml‘, ‘lake−manifest.json‘, ‘lean−toolchain‘: these are hash−verified, and any modification voids the entire run.
− No ‘axiom‘, no ‘admit‘, and no ‘sorry‘ left in your own proof. The certificate is checked for custom axioms and for ‘sorry‘; a run relying on either is discarded.

Rules

− Only read and write files in this working directory; the Lean project root is the current directory.
− You MUST use the lean−lsp MCP tools (mcp__lean−lsp__*) to check as you work; do not fall back to ‘lake env lean‘. ‘Bash(lake build:*)‘ is permitted and safe.
− Search before guessing: ‘lean_local_search‘ for local declarations, ‘lean_loogle‘ or ‘lean_leansearch‘ for Mathlib by type or description, ‘lean_state_search‘ for lemmas that close the current goal, ‘lean_multi_attempt‘ to test candidate tactics without editing the file, ‘lean_hover_info‘ for the type of an identifier. One well−chosen Mathlib lemma is often the whole proof.
− If the goal needs an argument about a family of indices a sum, a maximum, a minimum, a count, an ordering that is expected, and it is the point of the task: state the general claim as a helper lemma and prove it, rather than trying to close the goal with one tactic.
− Verify before you finish: ‘lean_diagnostic_messages‘ must show no errors and your ‘sorry‘ gone, and ‘lean_verify‘ on the declaration containing your goal must report axioms without ‘sorryAx‘.

Final Message

1. that the goal is closed, and how you proved it;
2. every helper lemma or ‘have‘ you added, with its statement;
3. the axiom list ‘lean_verify‘ reported for the declaration containing the goal;
4. if you could not close it: the goal state you are left with, and if you believe the target is **false** as stated the counterexample or obstruction, said plainly, instead of leaving a partial attempt in place.

C.4 BASE SKILL

This skill is shipped as a `SKILL.md` file that is installed into the sandbox, the agent reads it on its own once it judges the task to match. It fixes the framework the certificate must follow: the

`MILPObservationEquivalence` structure and its four conditions, the layout of the certificate file, and the trust requirements the proof must respect.

Base skill

Lean MILP Observation−Equivalence Standard

A certificate proves that two MILP formulations satisfy the four feasibility and objective non−increase conditions for minimization problems. It must use the project's ‘MILPObservationEquivalence‘ structure and must not invoke any optimality predicate.

When ‘map.md‘ and ‘map.py‘ are supplied, the parameter map is given, not derived: ‘paramMap‘ must implement that correspondence. The forward and backward variable maps remain yours to find.

‘MILPObservationEquivalence‘ at a glance

‘MILPObservationEquivalence F G‘ has exactly these nine fields:

− ‘paramMap : F.Params G.Params‘
− ‘fwd : (p : F.Params) F.Vars p G.Vars (paramMap p)‘
− ‘bwd : (p : F.Params) G.Vars (paramMap p) F.Vars p‘
− ‘fwd_feasible : p x, F.feasible p x G.feasible (paramMap p) (fwd p x)‘
− ‘bwd_feasible : p y, G.feasible (paramMap p) y F.feasible p (bwd p y)‘
− ‘objMap : ‘
− ‘objMap_mono : StrictMono objMap‘
− ‘fwd_obj_le : p x, F.feasible p x G.obj (paramMap p) (fwd p x) objMap (F.obj p x)‘
− ‘bwd_obj_le : p y, G.feasible (paramMap p) y objMap (F.obj p (bwd p y)) G.obj (paramMap p) y‘

‘objMap‘ converts the objective values of F into the scale of G. It is a single global function: it does not take the instance ‘p‘ as an argument, and the same function serves every instance. It must be strictly increasing, since it may only change the scale of the objective, never the ordering of solutions. Use the identity function when the two objectives already share a scale.

Both objective inequalities place ‘objMap‘ on the F side and leave ‘G.obj‘ bare on the other: the direction ‘G side objMap (F side)‘ must not be reversed, and ‘StrictMono‘ must not be weakened to ‘Monotone‘.

Do not add fields, optimality predicates, or optimal−solution correspondence theorems beyond the nine fields above.

File structure

Every proof file contains, in order:

1. ‘import Common‘, both formulation modules, and targeted Mathlib imports;
2. ‘open BigOperators Finset‘ only when needed;
3. a namespace matching the shared problem scope;
4. private helper definitions and lemmas, if needed;
5. the final definition whose exact type is ‘MILPObservationEquivalence A.formulation B.formulation‘.

Keep all helpers in the reformulation file. Use explicit field−by−field parameter maps when A and B use distinct parameter types, even if their fields look identical. Mark definitions ‘noncomputable‘ only when their construction requires it.

The A and B formulation files are independent specifications and are frozen after their first successful compilation. A proof may modify only ‘Reformulation.lean‘.

Proof guidance

Prove each feasibility implication directly from the corresponding formulation definition and assumptions. Prove each objective inequality with the exact ‘‘ direction shown above, applying ‘objMap‘ to the F−side objective value. Use local helper lemmas for algebraic or finite−sum rewrites, and expose required ‘NeZero‘ or finite−type instances from the parameter hypotheses rather than inventing assumptions.

Do not add any domain−specific lemma to this generic skill or template. The agent must derive all required facts from the supplied formulation descriptions and may place local helper lemmas only in Reformulation.lean.

Trust requirements

The certificate must compile with Lean and must not contain ‘sorry‘, ‘admit‘, ‘sorryAx‘, or custom axioms. Its endpoint must be exactly A to B: ‘MILPObservationEquivalence A.formulation B.formulation‘.

D BENCHMARK CONSTRUCTION

Our benchmark contains 15 MILP problems and 135 formulation pairs. Each problem is given one reference formulation together with nine candidate formulations, where each candidate is obtained by applying one of nine transformations to the reference, so that the same nine transformations occur in every problem. Of the 135 pairs, 105 preserve the optimal outcome of the reference under every parameter and carry the label true, and the remaining 30 carry the label false. This appendix

documents how the benchmark was built. Section D.1 states where the 15 problem types come from and how the reference formulation of each was obtained, and Section D.2 describes the nine transformations in detail.

D.1 Problem Types

The 15 problem types of our benchmark are listed in Table 5. Ten of them are taken from OPT-Engine (Chen et al., 2026), which provides canonical formulations of ten operations research problem classes. The remaining five types follow standard treatments of the corresponding problem classes (Vazirani, 2001; Nemhauser & Trotter, 1975; Irnich & Desaulniers, 2005; Smith, 1956).

Table 5: The 15 problem types of our benchmark and their sources.

Source	Problem type
(Chen et al., 2026)	Knapsack, bin packing, inventory planning, job shop scheduling, portfolio allocation, travelling salesman, air-pollution control, production planning, transportation, network flow
(Frieze & Jerrum, 1997)	Maximum k-cut, maximum satisfiability
(Nemhauser & Trotter, 1975)	Maximum independent set
(Irnich & Desaulniers, 2005)	Resource-constrained shortest path
(Smith, 1956)	Single-machine scheduling

D.2 Transformations

The nine transformations are applied to the reference formulation of every problem, so that each problem contributes nine pairs. They fall into three families: three change the variables, four change the constraints, and two change the objective. Seven of the nine preserve the optimal outcome of the reference under every parameter and are labelled true, and the remaining two change the optimal solution set and are labelled false. Table 6 lists the nine transformations, the change that each one makes, and the label of the resulting pair.

Table 6: The nine transformations applied to the reference formulation of every problem.

Transformation	Change	Label
Variable transformations		
Variable substitution	one variable is replaced by another	true
Variable disaggregation	one variable is replaced by several	true
Auxiliary variable	a variable is added	true
Constraint transformations		
Redundant constraint	a constraint implied by the others is added	true
Optimality-preserving cut	feasible points that are not optimal are removed	true
Symmetry-breaking cut	some optimal solutions are removed	true
Optimality-destroying cut	every optimal solution is removed	false
Objective transformations		
Affine transformation	the objective is rescaled and shifted	true
Optimal-set refinement	the objective is changed to select one optimum	false

We use the knapsack problem as the example for the nine transformations. Its reference formulation is

$$\max \sum_i v_i x_i \qquad \text{s.t.} \qquad \sum_i w_i x_i \le C, \qquad x \in \{0,1\}^n,$$

with parameters $w \in \mathbb{Z}^n_{>0}$, $v \in \mathbb{Z}^n_{>0}$ and $C \in \mathbb{Z}_{>0}$; we write $W = \sum_i w_i$ and $V = \sum_i v_i$ for the combined weight and the combined value. Two properties of the parameters are used below. First, every item fits into the empty backpack on its own, that is, $w_i \le C$, so taking a single item is feasible. Second, items 0 and 1 are the same piece of equipment from two brands, so $v_0 = v_1$ and $w_0 \le w_1$. The nine transformations are applied to this reference formulation, and each paragraph below states the variables it changes or the rows it adds, then the argument for its label.

Variable substitution. This transformation replaces one decision variable by another decision variable. The variant uses the complementary variables $y_i = 1 - x_i$, writes the objective over them as $V - \sum_i v_i y_i$, and has the feasible set

$$\Big\{ y \in \{0,1\}^n : \textstyle\sum_i w_i y_i \geq W - C \Big\}.$$

The map $x \mapsto y$ given by $y_i = 1 - x_i$ is a bijection between the two feasible sets, and the objective of the variant takes the same value at corresponding points. The two formulations therefore have the same optimal value for every parameter, and the pair is labelled true.

Variable disaggregation. This transformation replaces one decision variable by several decision variables. The quantity that is replaced is the unused capacity $s = C - \sum_i w_i x_i$, which is integer-valued and is therefore written in its binary expansion, using $T = \lfloor \log_2 C \rfloor + 1$ digits. The variant then has the feasible set

$$\Big\{ (x, b) \in \{0,1\}^n \times \{0,1\}^T : \textstyle\sum_i w_i x_i + \sum_{t<T} 2^t b_t = C \Big\}.$$

Since $0 \leq s \leq C < 2^T$, the digits represent the unused capacity uniquely, so dropping them is a bijection between the two feasible sets. The objective is unchanged, and the pair is labelled true.

Auxiliary variable. This transformation adds a decision variable that records an intermediate quantity, namely the capacity that is still available at every checkpoint, through the variables $y \in \mathbb{Z}^{n+1}$. The variant then has the feasible set

$$\Big\{ (x, y) \in \{0,1\}^n \times \mathbb{Z}^{n+1} : y_0 = C,\ y_{i+1} + w_i x_i = y_i \ (i < n),\ 0 \leq y_i \leq C \Big\}.$$

The rows determine y from x alone, namely $y_i = C - \sum_{j<i} w_j x_j$, so dropping y is a bijection between the two feasible sets. Because $w_i x_i \geq 0$ makes the sequence y_i non-increasing, the bounds $0 \leq y_i \leq C$ hold for every i exactly when $\sum_i w_i x_i \leq C$. The objective is unchanged, and the pair is labelled true.

Redundant constraint. This transformation adds a constraint that the constraints already present imply, so that the feasible set does not change. Every item weighs at least $\min_i w_i$, so $\min_i w_i \sum_i x_i \leq \sum_i w_i x_i \leq C$, and since $\sum_i x_i$ is a whole number the variant may add the row

$$\textstyle\sum_i x_i \leq K, \qquad K = \lfloor C / \min_i w_i \rfloor.$$

The added row holds at every feasible selection and for every legal parameter, so the two feasible sets coincide and the objective is unchanged. The pair is labelled true.

Optimality-preserving cut. This transformation adds a constraint that every optimal solution satisfies under every legal parameter, while some other feasible points violate it. Every item fits into the empty backpack on its own, so taking the single most valuable item is feasible and the optimal value is at least $V_{\text{best}} = \max_i v_i$. The variant adds the row

$$\textstyle\sum_i v_i x_i \geq V_{\text{best}}.$$

A selection that breaks this row has a total value below V_{best}, and a selection of value V_{best} exists, so such a selection is not optimal. This argument uses no property of the parameter, so for every legal parameter a selection that the row removes is not optimal, and every optimal selection satisfies the row. The optimal value is the same at every parameter, and the pair is labelled true.

Symmetry-breaking cut. This transformation adds a constraint that removes some optimal solutions while keeping others. Items 0 and 1 are the same piece of equipment from two brands, so $v_0 = v_1$ and $w_0 \leq w_1$, and the variant adds the row $x_1 \leq x_0$. A feasible selection with $x_1 = 1$ and $x_0 = 0$ can be replaced by the selection that swaps the two items; the swap stays feasible, because $w_0 \leq w_1$, and keeps the same value, because $v_0 = v_1$. For every legal parameter the variant therefore keeps a feasible selection of every value that the reference attains, in particular an optimum, although it does not keep every optimum. The optimal value is the same at every parameter, and the pair is labelled true.

Optimality-destroying cut. This transformation adds a constraint that no optimal solution satisfies. The variant asks the total value of the selection to be at least the weight capacity, by adding the row

$$\textstyle\sum_i v_i x_i \geq V_{\mathrm{req}}, \qquad V_{\mathrm{req}} = C.$$

For the supplied parameter values no selection reaches a total value of C, so the variant admits no feasible selection while the reference has an optimum. The optimal values of the two formulations therefore differ at those parameter values, and the pair is labelled false.

Affine transformation. This transformation changes the objective and leaves the feasible set unchanged, by replacing the objective $f(x) = \sum_i v_i x_i$ with $2f(x) + 3$:

$$\max \;\; 2\textstyle\sum_i v_i x_i + 3.$$

The constraint rows are the rows of the reference, so the two feasible sets coincide. A positive affine map of the objective preserves which selections are optimal and carries the optimal value along with it: whenever the reference has optimal value ψ, the variant has optimal value $2\psi + 3$. This holds at every legal parameter, and the pair is labelled true.

Optimal-set refinement. This transformation adds a second, lower-priority criterion to the objective, so that the optimal solution set is refined: some optimal solutions of the reference become suboptimal, while the selection that is best under the secondary criterion stays optimal. The variant maximizes

$$L\textstyle\sum_i v_i x_i + \sum_i 2^i x_i, \qquad L = 2^n,$$

whose first term is the objective of the reference weighted by $L = 2^n$ and whose second term is the secondary criterion, namely the index sum of the selected items. Because $\sum_i 2^i x_i \leq 2^n - 1 < L$, the second term cannot outweigh a difference in the first, so the variant orders the feasible selections by the value of the reference objective first and by the index sum second. The second term depends on the parameter, both through $L = 2^n$ and through which selections are optimal, so two legal parameter values with the same reference optimal value can have different variant optimal values. The optimal values of the two formulations are therefore not related by a single strictly increasing map, and the pair is labelled false.

E Experimental Details

E.1 Optimality-preserving cut in the knapsack problem

The first case is a narrowing of the feasible set that OBEquival certifies and that FLARE's constructive criterion does not. It is a knapsack pair in which the variant adds an optimality-preserving cut, that is, a row that every optimal solution satisfies. We write a pair as $\mathrm{p}N.x \to \mathrm{p}N.y$, where N numbers the problem and the letters identify the reference formulation and the variant.

A knapsack problem: a hiker packs a backpack for a three-day trip, item i has weight w_i and value v_i, and the combined weight of the selected items may not exceed the capacity C. Items 0 and 1 are the same piece of equipment from two brands, so $v_0 = v_1$ and $w_0 \leq w_1$; every item fits into the empty backpack on its own, $w_i \leq C$; and every value is positive, $v_i \geq 1$. The reference formulation selects the items to take,

$$\max \sum_i v_i x_i \qquad \text{s.t.} \qquad \sum_i w_i x_i \leq C, \qquad x \in \{0,1\}^n,$$

and the variant adds a single row, which asks the total value of the selection to be at least the value of the most valuable single item:

$$\begin{aligned} \max \; & \sum_i v_i x_i \\ \text{s.t.} \; & \sum_i w_i x_i \leq C, \quad \sum_i v_i x_i \geq V_{\mathrm{best}}, \quad x \in \{0,1\}^n, \\ & V_{\mathrm{best}} = \max_i v_i. \end{aligned}$$

The added row removes feasible selections, namely those whose total value is below V_{best}. The pair is labelled true, and the reason is that no removed selection is optimal: since every item fits into the empty backpack on its own, the single most valuable item is feasible, so the optimal value of the reference is at least V_{best}, and every optimal selection satisfies the added row.

The certificate that OBEquival produces for this pair is shown in part below. The parameter map is the identity on the parameters of the problem, and the box gives the remaining maps together with the proof that the objective map is strictly increasing; the lemma that the values attain their maximum, the lemmas that discharge the four conditions, and the declaration that assembles the certificate are omitted at the marked places.

Maps of the Observational Equivalence for reference formulation→variant formulation

```
-- ...: the lemma that the values attain their maximum is omitted

noncomputable def maxIndex (p : A.Params) : Fin p.n :=
  Classical.choose (exists_max_index p)

lemma maxIndex_spec (p : A.Params) :
    p.v (maxIndex p) = B.Vbest (paramMap p) := by
  exact Classical.choose_spec (exists_max_index p)

def unitVector (p : A.Params) (j : Fin p.n) : B.Vars (paramMap p) where
  x := fun i => if i = j then 1 else 0

-- forward map: keep a selection that satisfies the added row,
    otherwise
-- replace it by the selection that takes only the most valuable item
noncomputable def fwd (p : A.Params) (x : A.Vars p) : B.Vars (paramMap
    p) :=
  if B.Vbest (paramMap p) <= sum i : Fin p.n, p.v i * x.x i then
    { x := x.x }
  else
    unitVector p (maxIndex p)

-- the backward map and the objective map are the identity
def bwd (p : A.Params) (y : B.Vars (paramMap p)) : A.Vars p where
  x := y.x

def objMap : R -> R := fun r => r

lemma objMap_mono : StrictMono objMap := by
  intro a b hab
  exact hab

-- ...: the lemmas that discharge the four conditions, and the assembly
```

The backward map and the objective map are the identity, so neither a selection nor an objective value is translated between the two formulations. The forward map is the case split: a selection that satisfies the added row is kept, and a selection that violates it is replaced by the one that takes only the most valuable item, whose index the certificate computes first. The process proves this pair in one attempt, opening four "sorry" placeholders in the sketch and closing them in four sessions, for a total of 728 seconds and 2.9 million input tokens. Lean accepts the assembled certificate with no "sorry" placeholder and with the three standard axioms of Lean alone.

FLARE's criterion cannot certify this pair, and running FLARE on it reports the obstruction reproduced in the box below. Its criterion asks the value of every feasible selection of the reference to equal the value of the image of that selection under a single objective map shared by all parameters, where the image must be feasible in the variant.

FLARE's output on reference formulation→variant formulation

```
NOT REFORMULATION: Under the required identity parameter map, take n =
    2, C = 1, w(0) = w(1) = 1, and v(0) = v(1) = 1. These parameters
    satisfy every frozen assumption. In A, both the empty packing and
    either singleton packing are feasible, with objectives 0 and -1
    respectively. In B, feasibility forces exactly one item to be
    selected (the capacity permits at most one and hval requires at
    least one), so every feasible B point has objective -1. Thus
    fwd_obj would give objMap 0 = -1 and objMap (-1) = -1,
    contradicting StrictMono objMap since -1 < 0.
```

The witness is a legal parameter value at which no such map exists. With $n = 2$, $C = 1$ and $w_i = v_i = 1$ for both items, the reference admits the empty packing and either single-item packing, with values 0 and 1, while the variant admits exactly the single-item packings, because the capacity allows at most one item and the added row requires at least one; the run reports these values as 0 and -1, on the minimization scale on which its criterion is stated. Every feasible selection of the variant has value 1, so the equality forces the objective map to send both 0 and 1 to 1, contradicting the requirement that it be strictly increasing. The run accordingly reports the pair as not a reformulation, although the two formulations have the same optimal outcome at every legal parameter; this is the false negative of the constructive criterion that this case is meant to expose. The structural baseline rejects the pair as well, because the added row changes its constraint count, so a decision from structure loses this true pair for a reason that says nothing about the row.

E.2 FOOD-DISTRIBUTION PROBLEM IN FORMULATIONBENCH

Problem p20 of FormulationBench is a food-distribution problem. Let N be the set of nodes, partitioned into suppliers N_S, transshipment nodes N_T and camps N_B; let K be the set of commodities and L the set of nutrients; let $E_{ij} \in \{0,1\}$ indicate the arcs, tc_{ijk} the unit transport cost of commodity k on arc (i,j), pc_k the price per kilogram of commodity k, dem_j the demand at camp j, cap_t the handling capacity of transshipment node t, and nutval_{kl} and nutreq_l the nutrient content and requirement. The three node groups are disjoint, no arc enters a supplier and none leaves a camp, and the units used are compatible with positive values throughout. The reference formulation sends one flow variable F_{ijk} per arc, commodity and kilogram, together with the ration sizes R_k:

$$
\begin{aligned}
\min \quad & \sum_{k\in K} pc_k \sum_{s\in N_S}\sum_{j\in N} E_{S_s j} F_{S_s jk} \;+\; \sum_{i\in N}\sum_{j\in N}\sum_{k\in K} tc_{ijk} F_{ijk} \\
\text{s.t.} \quad & \sum_{i\in N} E_{iT_j} F_{iT_jk} = \sum_{i\in N} E_{T_j i} F_{T_j ik} && \forall\, j \in N_T,\ k \in K, \\
& \sum_{k\in K}\sum_{i\in N} E_{iT_t} F_{iT_tk} \le \mathrm{cap}_t && \forall\, t \in N_T, \\
& \sum_{i\in N} E_{iB_j} F_{iB_jk} \ge \mathrm{dem}_j R_k && \forall\, j \in N_B,\ k \in K, \\
& \sum_{k\in K} \mathrm{nutval}_{kl} R_k \ge \mathrm{nutreq}_l && \forall\, l \in L, \\
& F_{ijk} \in \mathbb{Z}_{\ge 0}, \quad R_k \ge 0,
\end{aligned}
$$

where S_s is supplier s and T_j, B_j are transshipment node and camp j. The variant replaces the flow on the arcs by amounts on routes: it writes the solution as x_{pk} on each simple path p that starts at a supplier and ends at a camp, and as y_{ck} on each simple cycle c, with the route costs $pCost_{pk} = \sum_{(i,j)\in p} tc_{ijk}$ and $cCost_{ck} = \sum_{(i,j)\in c} tc_{ijk}$ and the indicator $e_{jp} = 1$ for the paths

that end at camp j:

$$
\begin{aligned}
\min \quad & \sum_{p\in P}\sum_{k\in K} pCost_{pk}x_{pk} + \sum_{c\in C}\sum_{k\in K} cCost_{ck}y_{ck} + \sum_{k\in K} q_k \sum_{p\in P} x_{pk} \\
\text{s.t.} \quad & \sum_{k\in K}\Big(\sum_{p\in P}\big(\textstyle\sum_{i\in N} pE_{p,iT_t}\big)x_{pk} + \sum_{c\in C}\big(\sum_{i\in N} cE_{c,iT_t}\big)y_{ck}\Big) \le \mathrm{cap}_t \quad \forall\, t \in N_T, \\
& \sum_{p\in P} e_{jp}x_{pk} \ge \mathrm{dem}_j R_k \qquad \forall\, j \in N_B,\ k \in K, \\
& \sum_{k\in K} \mathrm{nutval}_{kl} R_k \ge \mathrm{nutreq}_l \qquad \forall\, l \in L, \\
& x_{pk}, y_{ck} \in \mathbb{Z}_{\ge 0}, \quad R_k \ge 0,
\end{aligned}
$$

where P is the set of the paths, C the set of the cycles, pE and cE their incidence indicators, and q_k the cost of the commodity bought at a supplier. The two formulations agree exactly when every nonnegative integral arc flow that is conserved at the transshipment nodes can be written as amounts on paths and cycles, which is the integral path-cycle decomposition. The pair p20.a→p20.b is labelled true.

FLARE's process gives one session the whole certificate. It produces a certificate that compiles and contains no placeholder, but the certificate does not establish the equivalence: the hard part enters as six custom axioms, one for each of the two maps and one for each of the three conditions that constrain them, together with the pair's parameter map. The box below reproduces the axiom check of that certificate and the closing message of the session.

Axiom check of the one-session certificate, and the session's closing message

```
propext
Classical.choice
Quot.sound
_private._AxiomCheck.0.WFP.bwd
_private._AxiomCheck.0.WFP.bwd_feas
_private._AxiomCheck.0.WFP.bwd_obj
_private._AxiomCheck.0.WFP.fwd
_private._AxiomCheck.0.WFP.fwd_feas
_private._AxiomCheck.0.WFP.fwd_obj

I could not complete a genuine axiom-free reformulation certificate.
    The current
certificate compiles and contains no placeholder, but it still relies
    on custom
axioms for the flow-decomposition maps and proofs, so it does not
    satisfy the
requested proof standard.
```

Our process fixes the order of the argument before any proof is written, and then writes the file along that order. The plan states the maps first, treats the four conditions in turn, and names the intermediate facts on which the proof depends. The box below reproduces its list of those facts and its treatment of the conditions.

Informal plan for p20.a→p20.b

```
# Proof plan: observation equivalence of the arc and route formulations

... the maps and the parameter correspondence are stated first ...

Integral path-cycle decomposition lemma. For a fixed commodity, suppose
     F is a nonnegative integral flow, is zero off the graph, and is
    conserved at every transshipment node. Then there are nonnegative
    integers x_p for every path p and y_c for every cycle c such that,
```

```
    for every ordered pair of nodes, F_ij = sum_p pE_pij x_p + sum_c
    cE_cij y_c.
1. Forward feasibility    2. Backward feasibility
3. Forward objective      4. Backward objective
Formalization-critical lemma list. The proof depends on the following
    claims, in this order: finiteness and correctness of the prescribed
     path/cycle enumeration; valid-path incidence balance at all
    transshipment nodes; exclusion of suppliers and camps from cycles;
    the integral path-cycle decomposition with edgewise reconstruction;
     camp-inflow reconstruction; transshipment-inflow reconstruction;
    supplier-outflow reconstruction; and transportation-cost expansion.
```

The sketch then writes the file along the same order, one declaration per claim, and leaves a hole wherever a claim is not immediate. The box below lists the declarations of the resulting file, in the order in which they appear.

Sketch for p20.a→p20.b

```
routeEnumeration_exists          -- the path and cycle enumeration is
    finite and valid
validPath_of_series
cycle_tail_injective
validCycle_of_series
positiveFlow_has_path_or_cycle   -- the support of a nonnegative flow
validPath_conservation_at_T
validPath_has_edge
validCycle_has_edge
natFlow_decomposition
flowDecomposition_exists         -- the integral path-cycle
    decomposition
chosenCoefficients
chosenCoefficients_spec          -- the extracted path and cycle
    coefficients
fwd, bwd                         -- the two maps
fwd_capacity, fwd_demand         -- forward feasibility
bwd_flow, bwd_capacity, bwd_demand, bwd_offedge   -- backward
    feasibility
...                              -- the objective conditions and the
    assembly
```

Seven “sorry” holes remain in that list, one per claim that the sketch could not close at once, and seven sessions close them, each in about five minutes; the assembly then rechecks the whole file, and the axiom check reports the three standard axioms of Lean and nothing else. The pair is proved in 2610 seconds and 6.5 million input tokens, and its certificate contains 1915 lines. The difference between the two runs is therefore neither the model nor the pair but the arrangement of the work. A session that receives the whole certificate can reach the goal by assuming it, and nothing in the file distinguishes that from a proof of it; a session that receives one open goal of a file whose other goals are already closed has to prove that goal, and the axiom check then confirms that it did.