# Towards Practical Classical Planning Compilations of Numeric Planning

**Luigi Bonassi**[1], **Francesco Percassi**[2*], **Enrico Scala**[1]

[1]Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Brescia, Italy
[2]School of Computing and Engineering, University of Huddersfield, United Kingdom
luigi.bonassi@unibs.it,f.percassi@hud.ac.uk,enrico.scala@unibs.it

## Abstract

It is well known that numeric planning can be made decidable if the domain of all numeric state variables is finite. This bounded formulation can be polynomially compiled into classical planning with Boolean conditions and conditional effects preserving the plan size exactly. However, it remains unclear whether this compilation has any practical utility. To explore this aspect, this work revisits the theoretical compilation framework from a practical perspective, focusing on the fragment of simple numeric planning. Specifically, we introduce three different compilations. The first, called *one-hot*, aims to systematise the current practice among planning practitioners of modelling numeric planning through classical planning. The other two, termed *binary* compilations, extend and specialise the logarithmic encoding introduced in previous literature. Our experimental analysis reveals that the overly complex logarithmic encoding can, surprisingly, be made practical with some representational expedients. Among these, the use of axioms is particularly crucial. Furthermore, we identify a class of mildly numeric planning problems where a classical planner, i.e., LAMA, when run on the compiled problem, is highly competitive with state-of-the-art numeric planners.

**Code** — https://github.com/LBonassi95/BitBlast

## Introduction

Automated planning is the field of Artificial Intelligence that studies how to develop intelligent systems that can automatically decide the best action to reach some goal (Ghallab, Nau, and Traverso 2016). From a computational perspective, automated planning is the problem of finding a sequence of actions that achieves a goal, given a formal and predictive specification of the system under consideration. Among the various formalisation variants appearing in the literature (Haslum et al. 2019), one that has recently gained significant attention is numeric planning (Fox and Long 2003; Leofante 2023; Shleyfman, Gnad, and Jonsson 2023; Bonassi, Gerevini, and Scala 2024; Helal and Lakemeyer 2024). In numeric planning, actions and goals are represented using numeric state variables and conditions, which has proven useful in several applications (Parkinson et al. 2012; Vallati et al. 2016; Kiam et al. 2020; Kouaiti et al. 2024).

*Corresponding Author.


Numeric planning is known to be undecidable in its general form (Helmert 2002), but decidability can be recovered (unsurprisingly) by making the domain of numeric variables finite. Recently, it has been shown that this restriction allows numeric planning to be compiled into classical planning with conditional effects via a compilation that is both polynomial and plan size-preserving (Gigante and Scala 2023). A measure of expressiveness in planning is whether a problem can be compiled into another without increasing the size of the plans (Nebel 2000). Indeed, compilations that fail to achieve this result are believed to be impractical.

However, it is unclear whether the theoretical framework by Gigante and Scala (2023) can truly make classical planners usable to solve numeric planning problems. Similar encodings, commonly known as bit-blasting, are frequently used in the formal methods field (Clarke, Kroening, and Yorav 2003; Bailleux and Boufkhad 2003; Bruttomesso et al. 2007; Vizel, Nadel, and Malik 2017). We argue that similar approaches can also be useful in planning; however, care must be taken to focus on scalable formulations that do not burden classical planners excessively.

To address this gap, we put the theoretical encoding of Gigante and Scala (2023) into practice. Our overall question is: how far can classical planners go? We begin with a base compilation that formalises the current practice of modelling numeric state variables in classical planning. In analogy to the SAT literature (Björk 2011; Miller, Gitina, and Becker 2011), we call this compilation ONE-HOT. Intuitively, ONE-HOT creates a Boolean variable for each possible value of a numeric variable, tracking their changes as actions are applied. This approach is expected to scale poorly because it requires a variable for each possible value in the numeric domain. Then, we present two compilations based on a logarithmic encoding of the numeric state variables. Such compilations start from the definition provided by Gigante and Scala (2023) but provide a concrete implementation for the arithmetic operation involved in simple numeric planning tasks. One formulation employs axioms, a planning language feature that enables the derivation of variables based on the state assignment. To the best of our knowledge, this is the first systematic study of the practical aspects of handling numeric information within classical planning.

We tested the compilations on the domains from the latest International Planning Competition (Taitler et al. 2024).

Our findings reveal a class of mildly numeric planning tasks where the axiom-based compilation paired with LAMA (Richter and Westphal 2010) yields results comparable to, and at times superior to, those obtained with state-of-the-art numeric planners.

## Background

This section introduces the input language for our compilations: numeric planning in a normalised form leveraged by the compilation process. We then outline classical planning with conditional effects and axioms, the target formalism, followed by a brief overview of logical circuit theory.

### Numeric Planning

A zero-restricted numeric planning task ($\text{RT}_0$) is defined as a tuple $\langle \mathcal{F}, \mathcal{N}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ where $\mathcal{F}$ and $\mathcal{N}$ are sets of Boolean and numeric variables, taking value in $\mathbb{B} = \{\bot, \top\}$ and $\mathbb{Z}$, respectively. $\mathcal{A}$ is a set of actions, $\mathcal{I}$ is the initial state and $\mathcal{G}$ is the goal state description. A state $s$ is a full assignment over $\mathcal{F}$ and $\mathcal{N}$ and, given $v \in \mathcal{F} \cup \mathcal{N}$, $s[v]$ denotes the value assumed by $v$ in $s$. Each state $s$ is partitioned into the Boolean and numeric component, i.e., $s = s_{\mathcal{F}} \cup s_{\mathcal{N}}$. Boolean conditions are of the form $\langle v = b \rangle$, where $v \in \mathcal{F}$ and $b \in \mathbb{B}$, while numeric conditions are $\langle v \geq 0 \rangle$, where $v \in \mathcal{N}$. Conditions, regardless of type, can form formulae $\phi$ using the syntax: $\phi := \top \mid \bot \mid \langle v \geq 0 \rangle \mid \langle v' = b \rangle \mid \phi \wedge \phi \mid \phi \vee \phi$, where $v \in \mathcal{N}$, $v' \in \mathcal{F}$, and $b \in \mathbb{B}$.

An action $a \in \mathcal{A}$ is defined as a pair $\langle \text{pre}(a), \text{eff}(a) \rangle$, where $\text{pre}(a)$ is a formula $\phi$, and $\text{eff}(a) = \text{eff}_{\mathcal{F}}(a) \cup \text{eff}_{\mathcal{N}}(a)$ is a set of Boolean and numeric effects. Boolean effects are of the form $\langle v := b \rangle$, where $v \in \mathcal{F}$ and $b \in \mathbb{B}$, while numeric effects are of the form $\langle v \mathrel{+}= q \rangle$, with $v \in \mathcal{N}$ and $q \in \mathbb{Z}$. Finally, the goal $\mathcal{G}$ is defined as a formula.

A state $s$ satisfies a formula $\phi$, written as $s \models \phi$, iff $s$ is a model for $\phi$. An action $a$ is applicable in $s$ iff $s \models \text{pre}(a)$, yielding a new state $s' = \gamma(s, a)$ where:

$$s'[v] = \begin{cases} b & \text{if } \langle v := b \rangle \in \text{eff}_{\mathcal{F}}(a) \\ s[v] + q & \text{if } \langle v \mathrel{+}= q \rangle \in \text{eff}_{\mathcal{N}}(a) \\ s[v] & \text{otherwise.} \end{cases}$$

A plan for $\Pi$ is a sequence of actions $\pi = \langle a_1, \ldots, a_n \rangle$ from $\mathcal{A}$. Let $\langle s_0, \ldots, s_n \rangle$ be the sequence of states s.t. $s_0 = \mathcal{I}$, $s_{i+1} = \gamma(s_i, a_{i+1})$ for each $i \in \{0, \ldots, n-1\}$, $\pi$ is valid iff for each $i \in \{0, \ldots, n-1\}$ $s_i \models \text{pre}(a_{i+1})$ and $s_n \models \mathcal{G}$.

In the following, for conciseness, we will use $v$ ($\neg v$) instead of $\langle v = \top \rangle$ and $\langle v := \top \rangle$ ($\langle v = \bot \rangle$ and $\langle v := \bot \rangle$).

We now provide a running example of an $\text{RT}_0$ to accompany the explanations of compilations.

**Example 1** (Running Example)**.** *This task involves a single numeric variable $v$, initialised to $-3$. An action increments $v$ by $1$ without any precondition, and the goal requires $v \geq 0$. Formally $\Pi = \langle \emptyset, \{v\}, \{a\}, \{v := -3\}, \langle v \geq 0 \rangle \rangle$, where $a = \langle \top, \{\langle v \mathrel{+}= 1 \rangle\} \rangle$.*

An $\text{RT}_0$ can capture simple numeric planning (SNP) (Scala et al. 2020), that is, the fragment of numeric planning in which the numeric conditions are defined as linear arithmetic expressions and the variables can be increased by a constant quantity. This is an immediate consequence of the compilation by Kuroiwa et al. (2022) from SNP to restricted numeric planning (Hoffmann 2003). Intuitively, the transformation from SNP to $\text{RT}_0$ consists of replacing each condition $\langle \xi \geq 0 \rangle$, where $\xi$ is a linear arithmetic expression over $\mathcal{N}$, with the condition $\langle z_\xi \geq 0 \rangle$, where $z_\xi$ is a novel numeric variable that is always kept to hold the value of $\xi$. Since numeric variables in SNP take value in $\mathbb{Q}$, when necessary, the transformation scales the planning task to remain in $\mathbb{Z}$.

### Classical Planning

We now present the target formalism for our compilations: classical planning with conditional effects (Mattmüller et al. 2018; Gerevini, Percassi, and Scala 2024) and axioms (Thiébaux, Hoffmann, and Nebel 2005; Speck et al. 2019), hereinafter simply *classical planning*. Formally, a classical planning task is defined as a tuple $\langle \mathcal{F}, \mathcal{X}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{F}$ is a set of Boolean variables, $\mathcal{X}$ is a set of axioms, $\mathcal{A}$ is a set of actions, $\mathcal{I}$ is the initial state, and $\mathcal{G}$ is a formula representing the goal. Similarly to numeric planning, a state $s$ is a full assignment over $\mathcal{F}$. In contrast, in classical planning, we introduce the notion of axioms and derived variables (also known as *derived predicates*). Axioms have the form $d \leftarrow \phi$ where $d$ is a derived variable and $\phi$ is a propositional formula over $\mathcal{F} \cup \mathcal{D}$, where $\mathcal{D} = \{d \mid d \leftarrow \phi \in \mathcal{X}\}$ is the set of all derived variables. Intuitively, derived variables are Boolean variables whose truth is determined from a state $s$ following the definition of axioms. An axiom $d \leftarrow \phi$ specifies that $d$ is derived to be true from a state $s$ if and only if we can prove that $s \models \phi$, possibly using other axioms from $\mathcal{X}$. For example, let $\mathcal{F} = \{a, b\}$, let $s$ be a state over $\mathcal{F}$, and let $\mathcal{X} = \{d_1 \leftarrow a, d_2 \leftarrow b, d_3 \leftarrow d_1 \wedge d_2\}$. The set of derived variables is $\mathcal{D} = \{d_1, d_2, d_3\}$, and the set of axioms $\mathcal{X}$ specify that $d_1$ is true iff $s \models a$, $d_2$ is true iff $s \models b$, and $d_3$ is true iff $d_1$ and $d_2$ hold in $s$. As we can see, determining the truth of a derived variable may require evaluating multiple axioms. If the set $\mathcal{X}$ of axioms is *stratified* (Thiébaux, Hoffmann, and Nebel 2005), then, given a state $s$, it is possible to efficiently and uniquely determine whether any derived variable $d \in \mathcal{D}$ holds in $s$. A direct consequence is that it is always possible to determine whether a formula $\phi$ over $\mathcal{F} \cup \mathcal{D}$ is satisfied by a state $s$. Therefore, we assume that $\mathcal{X}$ is stratified. In the rest of the paper, if a classical planning task has no axioms, we omit the set $\mathcal{X}$.

Derived variables can be used to specify the goal $\mathcal{G}$, which is a formula over $\mathcal{F} \cup \mathcal{D}$, and the model of actions. An action $a \in \mathcal{A}$ is a pair $\langle \text{pre}(a), \text{eff}(a) \rangle$ where $\text{pre}(a)$ is a formula over $\mathcal{F} \cup \mathcal{D}$ and $\text{eff}(a) = \text{eff}_{\mathcal{F}}(a)$ is a set of (Boolean) conditional effects. A conditional effect has the form $\phi \triangleright \langle v := b \rangle$ where $\phi$ is a formula over $\mathcal{F} \cup \mathcal{D}$, $v \in \mathcal{F}$, and $b \in \mathbb{B}$. Intuitively, a conditional effect specifies that the variable $v$ gets assigned to $b$ in the next state iff $\phi$ holds in the current state. More formally, an action $a$ is applicable in $s$ iff $s \models \text{pre}(a)$, yielding a new state $s' = \gamma(s, a)$ where:

$$s'[v] = \begin{cases} b & \text{if } \phi \triangleright \langle v := b \rangle \in \text{eff}_{\mathcal{F}}(a) \text{ and } s \models \phi \\ s[v] & \text{otherwise.} \end{cases}$$

For conciseness, multiple conditional effects with the same condition $\phi$ are denoted using the notation $\phi \triangleright e$ where $e$ is

a set of Boolean effects. As for numeric planning, a valid plan $\pi$ for $\Pi$ is a sequence of actions $\pi = \langle a_1, \ldots, a_n \rangle$ from $\mathcal{A}$ that is recursively applicable starting from $\mathcal{I}$ leading to a state satisfying $\mathcal{G}$.

## Arithmetic Circuits

Let $v$ be an integer. We denote its binary representation in two's complement with $\kappa$ bits as the bit vector $\mathbb{B}_\kappa(v) = \langle v_{\kappa-1}, \ldots, v_0 \rangle$, also denoted with $\mathbf{v}$. Here, each $v_i$ is a Boolean value such that $v_i = \top$ when the $i$-th bit is 1, and $v_i = \bot$ when it is 0. The most significant bit $v_{\kappa-1}$ indicates the sign of $v$: if $v_{\kappa-1} = \top$, $v$ is negative; otherwise, is non-negative. The range of integers that can be represented with $\kappa$ bits is $\mathbb{Z}_\kappa = \{v \in \mathbb{Z} \mid -2^{\kappa-1} \leq v \leq 2^{\kappa-1} - 1\}$.

The addition and subtraction of two integers $x$ and $y$ can be implemented using a logical circuit called full adder. This circuit processes the binary representations of $x$ and $y$ in two's complement form (Kroening and Strichman 2016). In the rest of the paper, we use the symbol $\oplus$ for the exclusive or, i.e., $x_i \oplus y_i = (x_i \wedge \neg y_i) \vee (\neg x_i \wedge y_i)$.

**Definition 1** (Full Adder)**.** *Let $\mathbf{x}$ and $\mathbf{y}$ be two $\kappa$-bits vector. A full adder is a logical circuit that performs the binary addition of $\mathbf{x}$ and $\mathbf{y}$. It produces a $\kappa$-bit sum vector $\mathbf{z} = adder(\mathbf{x}, \mathbf{y})$, where the $i$-th sum and carry bit for $i \in \{0, \ldots, \kappa - 1\}$ is defined as*

$$c_{-1} \leftrightarrow \bot \tag{1}$$
$$z_i \leftrightarrow (x_i \oplus y_i) \oplus c_{i-1} \tag{2}$$
$$c_i \leftrightarrow (x_i \wedge y_i) \vee (c_{i-1} \wedge (x_i \oplus y_i)). \tag{3}$$

Let $x$ and $y$ be two integers such that $x, y \in \mathbb{Z}_\kappa$, $\mathbf{x} = \mathbb{B}_\kappa(x)$, $\mathbf{y} = \mathbb{B}_\kappa(y)$, $\overline{\mathbf{z}} = \mathbb{B}_\kappa(x + y)$ and $\mathbf{z} = adder(\mathbf{x}, \mathbf{y})$. Then, if $x + y \in \mathbb{Z}_\kappa$, the following formula holds: $\bigwedge_{i=0}^{\kappa-1}(z_i \leftrightarrow \overline{z}_i)$. Otherwise, a representation error occurs when $x + y \notin \mathbb{Z}_\kappa$. This can be detected when the signs of the addends are the same but differ from the sign of the resulting sum. Particularly, an *underflow* (*overflow*) $x + y < \inf(\mathbb{Z}_\kappa)$ ($x + y > \sup(\mathbb{Z}_\kappa)$) happens when $x_{\kappa-1} \wedge y_{\kappa-1} \wedge \neg z_{\kappa-1}$ ($\neg x_{\kappa-1} \wedge \neg y_{\kappa-1} \wedge z_{\kappa-1}$).

# One-Hot Compilation

In this section, we formalise the practice of encoding numeric variables with finite domains into classical planning using a one-hot encoding. To generalise this approach, we introduce a compilation called ONE-HOT, which takes as input an $\text{RT}_0$ and a finite integer domain $\mathfrak{D} = \{l, l+1, \ldots, u\}$, where $l$ and $u$ are integers with $l < u$. For a numeric variable $v \in \mathcal{N}$, we define a set of Boolean variables corresponding to each possible value in the domain, represented as $\{v_i \mid i \in \mathfrak{D}\}$. These are referred to as *one-hot variables*. When $v = i$ for some $i \in \mathfrak{D}$, the variable $v_i$ is set to true, while all other variables referring to $v$ are set to false. Additionally, for each $v \in \mathcal{N}$, we define a corresponding *sign variable* denoted as $v_\geq$, set to true whenever $v \geq 0$ and false otherwise.

The following definition shows how to transform a formula featuring numeric conditions into a formula defined with only Boolean conditions. This transformation step is essential for encoding action preconditions and goals from numeric to classical planning.

**Definition 2.** *Let $\phi$ be a formula defined over $\mathcal{F} \cup \mathcal{N}$. We define the formula $\tau_{\text{O}}(\phi)$ expressed using only Boolean conditions, as the formula obtained by rewriting $\phi$ where every numeric condition $\langle v \geq 0 \rangle$ is replaced with $\langle v_\geq = \top \rangle$, where $v_\geq$ is the sign variable of $v$.*

Now, provided we have a method to determine the truth of a numeric condition using the corresponding sign variable, we will show how to update the one-hot and sign variables based on the numeric effects of actions. Since numeric effects may exceed the bounds of $\mathfrak{D}$, we also introduce another Boolean variable to detect potential overflows, i.e., $\uparrow$.

**Definition 3.** *Let $\Pi = \langle \mathcal{F}, \mathcal{N}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ be an $\text{RT}_0$, and let $\mathfrak{D}$ be a finite integer domain. Let $\langle v{+}{=}q \rangle$ be a numeric effect of some action $a \in \mathcal{A}$ such that $q \in \mathfrak{D}$. This effect is encoded as the following set $\text{EFF}_{\text{O}}(v, q)$ of conditional effects, involving only Boolean variables:*

$$\begin{aligned}
\text{EFF}_{\text{O}}(v, q) &= \text{ADD}_{\text{O}}(v, q) \cup \text{SIGN}(v, q) \cup \text{OF}_{\text{O}}(v, q), \text{ where} \\
\text{ADD}_{\text{O}}(v, q) &= \{v_i \rhd \{\neg v_i, v_{i+q}\} \mid i, i + q \in \mathfrak{D}\} \\
\text{SIGN}(v, q) &= \text{SIGN}^+(v, q) \cup \text{SIGN}^-(v, q) \\
\text{SIGN}^+(v, q) &= \{v_i \rhd v_\geq \mid i, i + q \in \mathfrak{D}, i < 0, i + q \geq 0\} \\
\text{SIGN}^-(v, q) &= \{v_i \rhd \neg v_\geq \mid i, i + q \in \mathfrak{D}, i \geq 0, i + q < 0\} \\
\text{OF}_{\text{O}}(v, q) &= \{v_i \rhd \uparrow \mid i \in \mathfrak{D}, i + q \notin \mathfrak{D}\}.
\end{aligned}$$

$\text{ADD}_{\text{O}}(v, q)$ captures the change of the value of $v$ by $q$ for each value assumed by $v$ in $\mathfrak{D}$. It includes a conditional effect to switch from $v_i$ ($v = i$) to $v_{i+q}$ ($v = i + q$), provided that $i + q$ is within $\mathfrak{D}$. $\text{SIGN}(v, q)$ handles the sign change effects, i.e., crossing from negative to non-negative or vice versa. Finally, $\text{OF}_{\text{O}}(v, q)$ captures the possibility that the value resulting from $i + q$ falls outside $\mathfrak{D}$.

Now, having a method to encode formulae and effects involving numeric variables, we are ready to formalise ONE-HOT. In the next definition, we assume that $\mathfrak{D}$ is sufficiently large to encompass all numeric constants present in the task.

**Definition 4** (ONE-HOT Compilation)**.** *Let $\Pi = \langle \mathcal{F}, \mathcal{N}, \mathcal{A}, \mathcal{I}_\mathcal{F} \cup \mathcal{I}_\mathcal{N}, \mathcal{G} \rangle$ be an $\text{RT}_0$ and $\mathfrak{D} = \{l, \ldots, u\}$ with $l, u \in \mathbb{Z}$ and $l < u$, a finite integer domain.* ONE-HOT *generates a classical planning task with conditional effects $\Pi_\mathfrak{D} = \langle \mathcal{F}', \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$ such that:*

$$\mathcal{F}' = \mathcal{F} \cup \{v_i \mid i \in \mathfrak{D}, v \in \mathcal{N}\} \cup \{v_\geq \mid v \in \mathcal{N}\} \cup \{\uparrow\}$$

$$\mathcal{A}' = \{a' \mid a \in \mathcal{A}\}$$

$$a' = \langle \tau_{\text{O}}(\text{pre}(a)) \wedge \neg \uparrow, \text{eff}_\mathcal{F}(a) \cup \bigcup_{\langle v+=q \rangle \in \text{eff}_\mathcal{N}(a)} \text{EFF}_{\text{O}}(v, q) \rangle$$

$$\mathcal{I}' = \mathcal{I}_\mathcal{F} \cup \mathcal{I}_\mathfrak{D} \cup \mathcal{I}_\geq \cup \{\uparrow := \bot\}$$

$$\mathcal{I}_\mathfrak{D} = \bigcup_{\substack{v \in \mathcal{N} \text{ s.t.} \\ q = \mathcal{I}_\mathcal{N}[v]}} (\{v_q := \top\} \cup \{v_i := \bot \mid i \in \mathfrak{D}, i \neq q\})$$

$$\mathcal{I}_\geq = \bigcup_{\substack{v \in \mathcal{N} \text{ s.t.} \\ \mathcal{I}_\mathcal{N}[v] \geq 0}} \{v_\geq := \top\} \cup \bigcup_{\substack{v \in \mathcal{N} \text{ s.t.} \\ \mathcal{I}_\mathcal{N}[v] < 0}} \{v_\geq := \bot\}$$

$$\mathcal{G}' = \tau_{\text{O}}(\mathcal{G}) \wedge \neg \uparrow .$$

Essentially, ONE-HOT modifies only the numeric component of $\Pi$. For actions, ONE-HOT leaves the Boolean component of both the preconditions and effects unchanged. The preconditions are manipulated using the function $\tau_{\text{O}}(\cdot)$, while each numeric effect is replaced with a set of conditional effects according to Definition 3.

Similarly, in the initial state, the Boolean variables are left unchanged. For each numeric variable $v$, the corresponding one-hot variable $v_q$ is set to true where $\mathcal{I}[v] = q$, while all other one-hot variables referring to $v$ are set to false (see $\mathcal{I}_{\mathfrak{D}}$). The sign variables are initialised according to the sign of the variables $\mathcal{N}$ in the initial state $\mathcal{I}$ (see $\mathcal{I}_{\geq}$).

**Example 2** (ONE-HOT Compilation)**.** *Let $\Pi$ be the task presented in Example 1, and let $\mathfrak{D} = \{-3, \dots, 0\}$.* ONE-HOT *generates the planning task $\Pi_{\mathfrak{D}} = \langle \mathcal{F}', \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$ such that $\mathcal{F}' = \{v_{-3}, v_{-2}, v_{-1}, v_0, v_{\geq}, \uparrow\}$, $\mathcal{I}' = \{v_{-3} := \top, v_{-2} := \bot, v_{-1} := \bot, v_0 := \bot, v_{\geq} := \bot\}$, as $\mathcal{I}[v] = -3$, $\mathcal{G}' = \tau_{\text{O}}(\langle v \geq 0 \rangle) \wedge \neg \uparrow = v_{\geq} \wedge \neg \uparrow$, and $\mathcal{A}' = \{\langle \neg \uparrow, \text{ADD}_{\text{O}}(v, 1) \cup \text{SIGN}(v, 1) \cup \text{OF}_{\text{O}}(v, 1) \rangle\}$, where:*

$$\begin{aligned} \text{ADD}_{\text{O}}(v, 1) = &\{v_{-3} \triangleright \{\neg v_{-3}, v_{-2}\}, v_{-2} \triangleright \{\neg v_{-2}, v_{-1}\} \\ &v_{-1} \triangleright \{\neg v_{-1}, v_0\}\} \\ \text{SIGN}(v, 1) = &\{v_{-1} \triangleright v_{\geq}\} \\ \text{OF}_{\text{O}}(v, 1) = &\{v_0 \triangleright \uparrow\}. \end{aligned}$$

**Theorem 1.** ONE-HOT *is sound, and for a sufficiently large $\mathfrak{D}$,* ONE-HOT *is complete.*

*Proof Sketch.* Let $\pi'$ be the plan for $\Pi_{\mathfrak{D}}$, where $\Pi_{\mathfrak{D}}$ is the classical encoding of $\Pi$ using ONE-HOT. By construction, there exists a plan $\pi$ that solves $\Pi$, where the actions in $\pi$ are identical to those in $\pi'$, but with their native numeric structure. Specifically, the Boolean one-hot variables encoding the $\mathcal{N}$ variables of $\Pi$ correctly represent the corresponding numeric values, and the sign variables ensure the satisfaction of the numeric conditions in which they are involved. Thus, if $\pi'$ solves $\Pi_{\mathfrak{D}}$, then $\pi$ solves $\Pi$. The reverse direction follows from similar reasoning. □

## Binary Compilations

It is easy to see that one-hot encoding may work well when the variables' domains are narrow but becomes impractical with large domains.

To address this issue, we revise the logarithmic compilation approach by Gigante and Scala (2023) for the specific case of $\text{RT}_0$. Each numeric variable is binarised using two's complement representation, and numeric conditions are then defined by focusing on the most significant bit of this representation, i.e., the sign bit.

We present two compilations for this revised approach: BLAST and an alternative, more compact representation that leverages axioms, that is $\text{BLAST}_{\mathcal{X}}$. All the following BLAST-based compilations take an $\text{RT}_0$ as input and represent all numeric variables using $\kappa \in \mathbb{N}$ bits, with $\kappa \geq 1$. Similarly to the approach taken by Gigante and Scala (2023), these compilations use a common schema for encoding preconditions and goals, which is provided here for clarity. However, each BLAST-based compilation offers a different planning encoding for the full adder when modelling numeric effects. BLAST encodes the full adder using only conditional effects, whereas $\text{BLAST}_{\mathcal{X}}$ also leverages axioms to modularise the computation of the full adder.

### BLAST Compilation

First, we focus on the new variables used to represent numeric variables in binary form. Given a numeric variable $v \in \mathcal{N}$, we denote by $\mathbf{v} = \langle v_{\kappa-1}, \dots, v_0 \rangle$ the Boolean variables representing the two's complement of $v$ using $\kappa$ bits. The most significant bit, i.e., $v_{\kappa-1}$, is the sign bit, and it will be used to convert a formula involving numeric and Boolean conditions into one that exclusively uses Boolean conditions.

**Definition 5.** *Let $\phi$ be a formula defined over $\mathcal{F} \cup \mathcal{N}$. We define the formula $\tau(\phi)$ expressed using only Boolean conditions, as the formula obtained by rewriting $\phi$ where every numeric condition $\langle v \geq 0 \rangle$ is replaced with $\langle v_{\kappa-1} = \bot \rangle$, where $v_{\kappa-1}$ is the most significant bit of the binary representation of $v$ using $\kappa$ bits.*

Regarding the effects, BLAST encodes each numeric effect into multiple conditional effects. Specifically, we encode the numeric effect $\langle v{+}{=}q \rangle$ of some action $a \in \mathcal{A}$, with conditional effects mimicking the full adder provided in Definition 1, and detect possible overflows. The Boolean variable to monitor overflow is denoted as $\uparrow$.

For conciseness, in the rest of the paper, we use $\phi \bowtie \langle v := b \rangle$ (with $b \in \mathbb{B}$) to denote the pair of conditional effects $\{\phi \triangleright \langle v := b \rangle, \neg\phi \triangleright \langle v := \neg b \rangle\}$.

**Definition 6.** *Let $\Pi = \langle \mathcal{F}, \mathcal{N}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ be an $\text{RT}_0$, and let $\kappa \in \mathbb{N}$ with $\kappa \geq 1$. Let $\langle v{+}{=}q \rangle$ be a numeric effect of some action $a \in \mathcal{A}$ such that $q \in \mathbb{Z}_\kappa$ and let $\mathbf{q} = \mathbb{B}_\kappa(q)$. This effect is encoded as the following set $\text{EFF}_{\text{B}}(v, q)$ of conditional effects, involving only Boolean variables:*

$$\begin{aligned} \text{EFF}_{\text{B}}(v, q) = &\text{ADD}_{\text{B}}(v, q) \cup \text{OF}_{\text{B}}(v, q), \textit{ where} \\ \text{ADD}_{\text{B}}(v, q) = &\{\mathcal{Z}_i \bowtie v_i \mid i \in \{0, \dots, \kappa - 1\}\} \\ \text{OF}_{\text{B}}(v, q) = &\{v_{\kappa-1} \wedge q_{\kappa-1} \wedge \neg \mathcal{Z}_{\kappa-1} \triangleright \uparrow, \\ &\neg v_{\kappa-1} \wedge \neg q_{\kappa-1} \wedge \mathcal{Z}_{\kappa-1} \triangleright \uparrow\} \\ \mathcal{Z}_i = &(v_i \oplus q_i) \oplus \mathcal{C}_{i-1} \\ \mathcal{C}_i = &\begin{cases} \bot & \textit{if } i = -1 \\ (v_i \wedge q_i) \vee (\mathcal{C}_{i-1} \wedge (v_i \oplus q_i)) & \textit{otherwise}. \end{cases} \end{aligned}$$

The set $\text{ADD}_{\text{B}}(v, q)$ includes two conditional effects for each bit, encoding the condition $\mathcal{Z}_i = (v_i \oplus q_i) \oplus \mathcal{C}_{i-1}$, which represents the $i$-th bit of the binary vector resulting from the sum of $\mathbf{v}$ and $\mathbf{q}$, as detailed in Equation 2. We use two conditional effects: one sets $v_i$ to true when $\mathcal{Z}_i$ holds, and the other sets it to false otherwise.

The set $\text{OF}_{\text{B}}(v, q)$ handles potential overflow resulting from the numeric effect $\langle v{+}{=}q \rangle$ by using a formula that accounts for the two possible sources of representation error.

Note that the encoding of the effect does not explicitly represent the carry bits involved in the $\mathcal{Z}_i$ formula. Instead, these carry bits are defined recursively through Equation 3, which reconstructs the $i$-th carry bit $\mathcal{C}_i$ based on the sums of the previous $i - 1$ bits.

We are now ready to formalise BLAST. In the following definition, we assume that $\kappa$ is sufficiently large to represent all numeric constants present in the problem.

**Definition 7** (BLAST Compilation)**.** *Let* $\Pi = \langle \mathcal{F}, \mathcal{N}, \mathcal{A}, \mathcal{I}_{\mathcal{F}} \cup \mathcal{I}_{\mathcal{N}}, \mathcal{G} \rangle$ *be an* $\text{RT}_0$ *and* $\kappa \in \mathbb{N}$ *with* $\kappa \geq 1$*,* BLAST *generates a classical planning task with conditional effects* $\Pi_\kappa = \langle \mathcal{F}', \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$*, where:*

$$\mathcal{F}' = \mathcal{F} \cup \{v_i \mid i \in \{0, \dots, \kappa - 1\}, v \in \mathcal{N}\} \cup \{\uparrow\}$$
$$\mathcal{A}' = \{a' \mid a \in \mathcal{A}\}$$
$$a' = \langle \tau(\text{pre}(a)) \wedge \neg \uparrow, \text{eff}_{\mathcal{F}}(a) \cup \bigcup_{\langle v \mathrel{+}= q \rangle \in \text{eff}_{\mathcal{N}}(a)} \text{EFF}_{\text{B}}(v, q) \rangle$$
$$\mathcal{I}' = \mathcal{I}_{\mathcal{F}} \cup \mathcal{I}'_{\mathcal{N}} \cup \{\uparrow := \bot\}$$
$$\mathcal{I}'_{\mathcal{N}} = \bigcup_{\substack{v \in \mathcal{N} \text{ s.t.} \\ \mathbf{q} = \mathbb{B}_\kappa(\mathcal{I}_{\mathcal{N}}[v])}} \{v_{\kappa-1} := q_{\kappa-1}, \dots, v_0 := q_0\}$$
$$\mathcal{G}' = \tau(\mathcal{G}) \wedge \neg \uparrow .$$

In BLAST, actions are transformed using a compilation similar to ONE-HOT, except that we use Definition 5 to manipulate the action preconditions, i.e., $\tau(\cdot)$, and Definition 6, to encode the numeric effects, i.e., $\text{EFF}_{\text{B}}(\cdot)$.

In the initial state, we maintain the same assignments for the variables $\mathcal{F}$ while, given $v \in \mathcal{N}$, we initialise its binary representation according to $\mathbb{B}_\kappa(\mathcal{I}_{\mathcal{N}}[v])$ (see $\mathcal{I}'_{\mathcal{N}}$).

**Theorem 2.** BLAST *is sound, and for a sufficiently large* $\kappa$*,* BLAST *is complete.*

*Proof Sketch.* The proof follows from Theorem 3 by Gigante and Scala (2023) and the observation that all numeric effects progress the numeric variables in a manner consistent with the corresponding arithmetic operations. Specifically, for each numeric variable, there are two possible cases to consider: the variable is unaffected or affected by the action. In the first case, it is straightforward to see that the compilation preserves the frame axiom. In the second case, where the variable is modified, conditional effects ensure the correct computation of a full adder. □

**Example 3.** *We consider the* $\text{RT}_0$ $\Pi$ *provided in Example 1. Let* $\kappa = 3$*,* BLAST *gives us* $\Pi_\kappa = \langle \mathcal{F}', \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$ *such that* $\mathcal{F}' = \{v_2, v_1, v_0\} \cup \{\uparrow\}$*,* $\mathcal{I}' = \{v_2 := \top, v_1 := \bot, v_0 := \top, \uparrow := \bot\}$*, as* $\{v_2, v_1, v_0\}$ *encodes* $v = -3$ *in* $\mathcal{I}'$*, and* $\mathcal{G}' = \tau(\mathcal{G}) \wedge \neg \uparrow = \neg v_2 \wedge \neg \uparrow$*. Regarding the actions,* $\mathcal{A}' = \{a' = \langle \neg \uparrow, \text{ADD}_{\text{B}}(v, 1) \cup \text{OF}_{\text{B}}(v, 1) \rangle\}$*. For* $\text{ADD}_{\text{B}}(v, 1)$*, we encode* $\langle v \mathrel{+}= 1 \rangle$ *by resorting to* $\mathbb{B}_3(1) = \langle q_2, q_1, q_0 \rangle = \langle \bot, \bot, \top \rangle$*. Then, we enumerate all the formulae* $\mathcal{Z}_i$ *for each bit according to Definition 6:*

$$\mathcal{Z}_0 = (v_0 \oplus q_0) \oplus \mathcal{C}_{-1}$$
$$\mathcal{Z}_1 = (v_1 \oplus q_1) \oplus \mathcal{C}_0 = (v_1 \oplus q_1) \oplus \overbrace{((v_0 \wedge q_0) \vee (\mathcal{C}_{-1} \wedge (v_0 \oplus q_0)))}^{\mathcal{C}_0}$$
$$\mathcal{Z}_2 = (v_2 \oplus q_2) \oplus \mathcal{C}_1 = (v_2 \oplus q_2) \oplus ((v_1 \wedge q_1) \vee (\mathcal{C}_0 \wedge (v_1 \oplus q_1))) =$$
$$= (v_2 \oplus q_2) \oplus \overbrace{((v_1 \wedge q_1) \vee (\underbrace{((v_0 \wedge q_0) \vee (\mathcal{C}_{-1} \wedge (v_0 \oplus q_0)))}_{\mathcal{C}_0} \wedge (v_1 \oplus q_1)))}^{\mathcal{C}_1}.$$

*For* $\text{OF}_{\text{B}}(v, 1)$*, we enumerate the logical conditions for detecting an overflow according to Definition 6.*

*After simplifications, we obtain the following effects:*

$$\text{ADD}_{\text{B}}(x, 1) = \{\overbrace{\neg v_0 \bowtie v_0}^{\textit{bit 0}}, \overbrace{v_0 \oplus v_1 \bowtie v_1}^{\textit{bit 1}}, \overbrace{v_2 \oplus (v_0 \wedge v_1) \bowtie v_2}^{\textit{bit 2}}\}$$
$$\text{OF}_{\text{B}}(v, 1) = \{\neg v_2 \wedge (v_2 \oplus (v_0 \wedge v_1)) \triangleright \uparrow\}.$$

BLAST enables using any classical planner supporting conditional effects to solve a numeric task. However, BLAST has a major shortcoming: the set of conditional effects $\text{EFF}_{\text{B}}(v, q)$ encoding $\langle v \mathrel{+}= q \rangle$ includes formulae whose size grows with number of bits $\kappa$. This is because, as we can partially observe in Example 3, the formula capturing the $\mathcal{C}_{i-1}$ carry appearing in the condition of the $i$-th pair of conditional effects $(v_i \oplus q_i) \oplus \mathcal{C}_{i-1} \bowtie v_i$ contains the formulae $\mathcal{C}_{i-2}, \dots, \mathcal{C}_{-1}$ associated with the previous carry bits. As $\kappa$ grows, the formulae in the conditional effects can become large enough to limit the scalability of classical planners.

To overcome this limitation, we provide an alternative encoding that compactly simulates the two's complement sum between two integers by using the power of axioms and derived variables. Intuitively, at the price of adding a few more (derived) variables, axioms let us evaluate the outcome of the $i$-th bit without explicitly involving all previous bits.

## BLAST$_{\mathcal{X}}$ Compilation

Axioms and derived variables, as also observed in the literature (Thiébaux, Hoffmann, and Nebel 2005; Borgwardt et al. 2021; Bonassi et al. 2023), can compactly encode complex properties, thus simplifying the action model and goal specifications. In our case, we use axioms to deal with the main source of complexity of BLAST: the set of conditional effects $\text{EFF}_{\text{B}}(v, q)$. Specifically, given a numeric effect $\langle v \mathrel{+}= q \rangle$, we use axioms and derived variables to compactly define a set of conditional effects $\text{EFF}_{\mathcal{X}}(v, q)$ that mimic the full adder provided in Definition 1 and detect overflow.

**Definition 8.** *Let* $\Pi = \langle \mathcal{F}, \mathcal{N}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ *be an* $\text{RT}_0$*, and let* $\kappa \in \mathbb{N}$ *with* $\kappa \geq 1$*. Let* $\langle v \mathrel{+}= q \rangle$ *be a numeric effect of some action* $a \in \mathcal{A}$ *such that* $q \in \mathbb{Z}_\kappa$*, and let* $\mathbf{q} = \mathbb{B}_\kappa(q)$*. This effect is encoded using a set of axioms* $\mathcal{X}_{v,q}$ *and a set of conditional effects* $\text{EFF}_{\mathcal{X}}(v, q)$ *defined as follows:*

$$\mathcal{X}_{v,q} = \{c_{-1}^{v,q} \leftarrow \bot\} \cup \bigcup_{i=0}^{\kappa-1} \{z_i^{v,q} \leftarrow (v_i \oplus q_i) \oplus c_{i-1}^{v,q}\} \cup \bigcup_{i=0}^{\kappa-1} \{c_i^{v,q} \leftarrow (v_i \wedge q_i) \vee (c_{i-1}^{v,q} \wedge (v_i \oplus q_i))\}$$
$$\text{EFF}_{\mathcal{X}}(v, q) = \text{ADD}_{\mathcal{X}}(v, q) \cup \text{OF}_{\mathcal{X}}(v, q), \textit{ where}$$
$$\text{ADD}_{\mathcal{X}}(v, q) = \{z_i^{v,q} \bowtie v_i \mid i \in \{0, \dots, \kappa - 1\}\}$$
$$\text{OF}_{\mathcal{X}}(v, q) = \{v_{\kappa-1} \wedge q_{\kappa-1} \wedge \neg z_{\kappa-1}^{v,q} \triangleright \uparrow, \neg v_{\kappa-1} \wedge \neg q_{\kappa-1} \wedge z_{\kappa-1}^{v,q} \triangleright \uparrow\}.$$

Note that the set of axioms $\mathcal{X}_{v,q}$ compactly captures the propositional formulae $\mathcal{Z}_i$ and $\mathcal{C}_i$ defined in Definition 6.

We now present BLAST$_{\mathcal{X}}$. Essentially, BLAST$_{\mathcal{X}}$ works exactly as BLAST; the only difference is that BLAST$_{\mathcal{X}}$ uses

the axioms and conditional effects provided in Definition 8 to encode numeric effects.

**Definition 9** (BLAST$_{\mathcal{X}}$). *Let $\Pi = \langle \mathcal{F}, \mathcal{N}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ be an* RT$_0$ *and $\kappa \in \mathbb{N}$ with $\kappa \geq 1$,* BLAST$_{\mathcal{X}}$ *generates a classical planning task with conditional effects and axioms $\Pi_\kappa = \langle \mathcal{F}', \mathcal{X}, \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$ where $\mathcal{F}'$, $\mathcal{I}'$, $\mathcal{G}'$ are defined as in* BLAST*, and axioms and actions are defined as:*

$$\mathcal{X} = \bigcup_{a \in \mathcal{A}} \bigcup_{\langle v \mathrel{+}= q \rangle \in \text{eff}_{\mathcal{N}}(a)} \mathcal{X}_{v,q}$$

$$\mathcal{A}' = \{a' \mid a \in \mathcal{A}\}$$

$$a' = \langle \tau(\text{pre}(a)) \wedge \neg \uparrow, \text{eff}_{\mathcal{F}}(a) \cup \bigcup_{\langle v \mathrel{+}= q \rangle \in \text{eff}_{\mathcal{N}}(a)} \text{EFF}_{\mathcal{X}}(v, q) \rangle.$$

From Definition 9, it is easy to see that for any RT$_0$ $\Pi$ and any $\kappa$, both BLAST$_{\mathcal{X}}$ and BLAST produce equivalent compiled problems.

**Theorem 3.** BLAST$_{\mathcal{X}}$ *is sound, and for a sufficiently large $\kappa$,* BLAST$_{\mathcal{X}}$ *is complete.*

*Proof Sketch.* For an RT$_0$ $\Pi$, BLAST and BLAST$_{\mathcal{X}}$ yield equivalent compiled problems. Therefore, the thesis holds by Theorem 2. □

**Example 4.** *Let $\Pi$ be the problem of Example 1. Let $\kappa = 3$,* BLAST$_{\mathcal{X}}$ *gives us $\Pi_\kappa = \langle \mathcal{F}', \mathcal{X}, \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$ such that $\mathcal{F}', \mathcal{I}', \mathcal{G}'$ are as for* BLAST*, and*

$$\mathcal{X} = \{c_{-1}^{v,1} \leftarrow \bot,$$
$$c_0^{v,1} \leftarrow (v_0 \wedge q_0) \vee (c_{-1}^{v,1} \wedge (v_0 \oplus q_0)) = v_0 \vee (c_{-1}^{v,1} \wedge \neg v_0),$$
$$c_1^{v,1} \leftarrow (v_1 \wedge q_1) \vee (c_0^{v,1} \wedge (v_1 \oplus q_1)) = c_0^{v,1} \wedge v_1,$$
$$c_2^{v,1} \leftarrow (v_2 \wedge q_2) \vee (c_1^{v,1} \wedge (v_2 \oplus q_2)) = c_1^{v,1} \wedge v_2,$$
$$z_0^{v,1} \leftarrow (v_0 \oplus q_0) \oplus c_{-1}^{v,1} = \neg v_0 \oplus c_{-1}^{v,1},$$
$$z_1^{v,1} \leftarrow (v_1 \oplus q_1) \oplus c_0^{v,1} = v_1 \oplus c_0^{v,1},$$
$$z_2^{v,1} \leftarrow (v_2 \oplus q_2) \oplus c_2^{v,1} = v_2 \oplus c_1^{v,1}\}$$

$\mathcal{A}' = \{a' = \langle \neg \uparrow, \text{ADD}_{\mathcal{X}}(v,1) \cup \text{OF}_{\mathcal{X}}(v,1) \rangle\}$*, where*

$$\text{ADD}_{\mathcal{X}}(v,1) = \{z_0^{v,1} \bowtie v_0, z_1^{v,1} \bowtie v_1, z_2^{v,1} \bowtie v_2\}$$
$$\text{OF}_{\mathcal{X}}(v,1) = \{\neg v_2 \wedge z_2^{v,1} \triangleright \uparrow\}.$$

## Experimental Analysis

The experimental analysis assesses the capability of classical planners in solving numeric planning tasks and compares their results with those of state-of-the-art numeric planners.

We aim to investigate the extent to which classical planners can perform when using the different compilations and determine which encoding performs best in practice. Therefore, we have implemented the compilations in Python using the unified planning library (Micheli et al. 2025).

As benchmarks, we considered all the SNP domains from the latest International Planning Competition (Taitler et al. 2024), except SETTLERS. For SETTLERS, we used the SNP formulation from Scala et al. (2020), where all numeric variables are initialised in the initial state. In the original formulation, some variables were undefined in the initial state and initialised by actions, which makes the problem not SNP. Currently, we do not support tasks that are neither SNP nor involve undefined numeric variables. Similarly to what was done by Cardellini, Giunchiglia, and Maratea (2024), and to understand the behaviour of the compilations based on the structure of numeric problems, we considered a measure of the "numericity" of each RT$_0$ $\Pi$ calculated as $\mathfrak{N}(\Pi) = \frac{|\mathcal{N}|}{|\mathcal{N}|+|\mathcal{F}|}$. When $\mathfrak{N}(\Pi) > 0.5$, $\Pi$ is *strongly numeric* (SN) as there are more numeric variables than Boolean ones; otherwise, it is *mildly numeric* (MN).

As classical planners (the target of our compilation), we considered several options from previous competitions, focusing on LAMA (Richter and Westphal 2010) (specifically, stopping at the first solution, denoted as LAMA-FIRST), as it is a planner that supports axioms and also the best-performing system over our instances.

On the numeric side, we tested a wide range of different planners—PATTY (Cardellini, Giunchiglia, and Maratea 2024), ENHSP (Scala et al. 2020), and NLM-CUTPLAN (NLM) (Kuroiwa, Shleyfman, and Beck 2023b)—selecting a high-performing configuration for each. For PATTY, we used the default configuration. For ENHSP, we adopted the M$(3h||3n)$ configuration (https://github.com/hstairs/jpddlplus/tree/socs24-width-and-mq) from the work by Chen and Thiébaux (2024). For NLM, we used the SAT2 configuration (Kuroiwa, Shleyfman, and Beck 2022; Shleyfman, Kuroiwa, and Beck 2023), which is a lazy greedy best-first search paired with an admissible heuristic.

In our experiments, each test run with a classical planner involves a compilation method and a planning task from our benchmark suite. Specifically, each run first normalises the input SNP task into an RT$_0$, then compiles it, and finally provides the output to the selected off-the-shelf planner. Since all compilations require the size of the finite integer domain—such as $\kappa$ bits for the BLAST-based compilations and the range $\mathfrak{D}$ for ONE-HOT —we had to choose the appropriate size for each domain. For each domain, we identified the largest constant $K$ from the actions, initial state, and goal description, setting $\kappa = \lceil \log_2(K) \rceil + 1$ for all compiled planning tasks. To ensure a fair comparison, ONE-HOT used $\mathfrak{D} = \mathbb{Z}_\kappa$, while numeric planners were evaluated on the original numeric instances.

We allocate a budget of 1,800 seconds of runtime and 8 GB of memory (for normalisation, compilation, and solving). The experiments were conducted on an Intel Xeon Gold 6140M CPU with a clock speed of 2.30 GHz.

Table 1 shows the coverage (number of solved instances) achieved by LAMA-FIRST on instances compiled by ONE-HOT (O), BLAST (B) and BLAST$_{\mathcal{X}}$ (B$_{\mathcal{X}}$), compared to PATTY (P), ENHSP with the configuration M$(3h||3n)$ (M) and NLM with the configuration SAT2 (S). The table is divided into two subtables, each corresponding to a class of domains: SN and MN. We list the number of bits used to represent the numeric variables ($\kappa$) and the average numericity for each domain ($\mathfrak{N}$). As the table shows, BLAST$_{\mathcal{X}}$ outperforms the other compilations, achieving the highest coverage across all domains. This finding suggests that using axioms

| Domain (#) | $\mathfrak{N}$ | $\kappa$ | LAMA-FIRST O | LAMA-FIRST B | LAMA-FIRST $B_{\mathcal{X}}$ | S | P | M |
|---|---|---|---|---|---|---|---|---|
| BGrouping (20) | 1.0 | 8 | 2 | 2 | 2 | 0 | **20** | 16 |
| Counters (20) | 1.0 | 8 | 6 | 5 | 5 | 12 | **20** | 10 |
| Watering (20) | 1.0 | 9 | 0 | 0 | 2 | 19 | 6 | **20** |
| Farmland (20) | 1.0 | 15 | 0 | 0 | 4 | 15 | **20** | **20** |
| MTrader (20) | 0.94 | 15 | 0 | 0 | 0 | 2 | 7 | **20** |
| Sailing (20) | 0.88 | 12 | 0 | 0 | 5 | 10 | 19 | **20** |
| Pathways (20) | 0.7 | 15 | 14 | 14 | 14 | 1 | **20** | 3 |
| Sugar (20) | 0.64 | 7 | 15 | 12 | 11 | 6 | **20** | 13 |
| $\Sigma_{\text{SN}}$ (160) | — | — | 37 | 33 | 43 | 65 | **132** | 122 |
| Settlers (20) | 0.41 | 5 | 14 | **15** | **15** | 2 | *n/a* | 2 |
| Expedition (20) | 0.39 | 11 | 0 | 0 | 3 | 4 | 3 | **9** |
| HPower (20) | 0.32 | 17 | 0 | 0 | 1 | 18 | **20** | **20** |
| Rovers (20) | 0.12 | 8 | 6 | 4 | 15 | 8 | **18** | 16 |
| MPrime (20) | 0.08 | 5 | **20** | **20** | **20** | 12 | 17 | 19 |
| Delivery (20) | 0.02 | 6 | 19 | **20** | **20** | 9 | 5 | **20** |
| $\Sigma_{\text{MN}}$ (120) | — | — | 59 | 59 | 74 | 53 | 63 | **86** |
| $\Sigma$ (280) | — | — | 96 | 92 | 117 | 118 | 195 | **208** |

Table 1: Coverage. Bolds is for the overall best, and underline is the best among compilations. *n/a* denotes that the planner is incompatible with the domain.

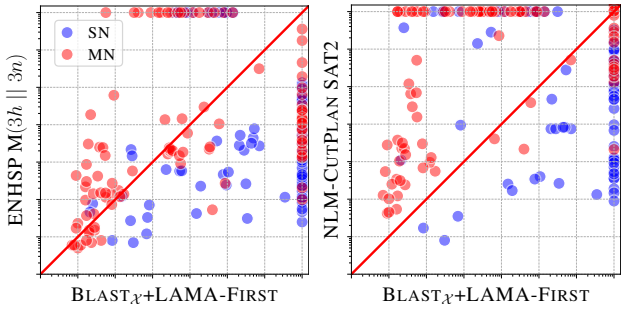


Figure 1: Expanded nodes scatter plot. The points above the bisector are in favour of BLAST$_{\mathcal{X}}$. Points on the margins represent unsolved instances.

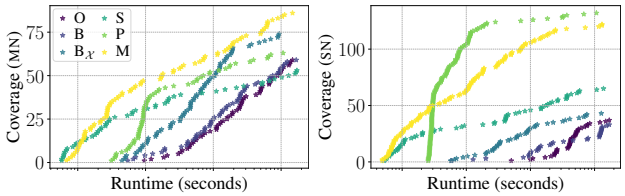


Figure 2: Coverage over time for MN (left) and SN (right).

is highly advantageous, even though the planner must evaluate numerous axioms to update the bits for each numeric variable at every search step. Interestingly, ONE-HOT and BLAST achieve comparable coverage.

Regarding the comparison between compilation-based and native systems, it is noteworthy that while BLAST$_{\mathcal{X}}$ performs poorly in strongly numeric domains (the top part of the table), its performance in MN is comparable to, and at times even exceeds, that of the numeric planners we tested. Indeed, BLAST$_{\mathcal{X}}$ is the runner-up in the MN tasks. In these domains, the gap between BLAST$_{\mathcal{X}}$ and M$(3h||3n)$ is much smaller compared to the SN domains. Specifically, and somewhat unsurprisingly, BLAST$_{\mathcal{X}}$ excels in the DELIVERY domain, which has only a few numeric variables. Unexpectedly, it also excels in SETTLERS, solving three-fourths of the instances. Although SETTLERS is a challenging numeric planning problem, the Boolean structure provides sufficient information for LAMA-FIRST to better guide the search.

Figure 2 shows the coverage over time for all systems, separately for SN and MN domains. Focusing on MN domains, naturally, compilation-based approaches are slower than native numeric planners due to the overhead of the compilation time. Indeed, BLAST$_{\mathcal{X}}$ becomes the runner-up performer after roughly 200 seconds. We also collected data on the number of expanded nodes by LAMA-FIRST combined with BLAST$_{\mathcal{X}}$ and numeric planners based on forward search, i.e., M$(3h||3n)$ and NLM SAT2 (Figure 1). This is a measure of the informedness of the heuristics. For SN instances, both numeric planners expand fewer nodes, while for the MN instances, LAMA-FIRST expands fewer or comparable nodes. The separation between SN and MN instances is even more pronounced in the comparison with NLM SAT2. The results suggest that a simple criterion based on the numericity of the planning task to be solved can guide the choice between using numeric or classical planners.

We also tested our compilations with SYMK (Speck et al. 2019), a state-of-the-art optimal planner supporting axioms. Overall, SYMK solved 23 problems with BLAST$_{\mathcal{X}}$, 15 with BLAST, and 0 with ONE-HOT. For comparison, the optimal configuration ORBIT for NLM solves a total of 86 problems. It is worth noting that with compilation approaches, the optimal plans may vary depending on the choice of $\kappa$. Therefore, these results should be seen as indicative rather than a fair comparison for optimal planning.

## Conclusions and Future Work

This paper studies practical compilations from numeric to classical planning and presents three encodings, two of which use logarithmic representations for numeric variables. Our experiments positively address whether there are practical benefits from the theoretical results provided by Gigante and Scala (2023). Specifically, we show that the compilation closest to the theory has limited scalability. In contrast, a refined encoding of the full adder through axioms enables classical planners to compete with native numeric planners in some domains. We identify several avenues for future work: first, we want to study novel numeric heuristics leveraging our compilations; second, we plan to combine our approach with an automated way for extracting bounds from a numeric planning task (Kuroiwa, Shleyfman, and Beck 2023a). Finally, we would like to explore practical compilations for more expressive fragments of numeric planning.

## Acknowledgements

Luigi Bonassi and Enrico Scala were supported by PRIN project RIPER (No. 20203FFYLK), project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU, and specifically by the project NEACD: Neurosymbolic Enhanced Active Cyber Defence (CUP J33C22002810001). Francesco Percassi was supported by a UKRI Future Leaders Fellowship [grant number MR/T041196/1].

## References

Bailleux, O.; and Boufkhad, Y. 2003. Efficient CNF Encoding of Boolean Cardinality Constraints. In *CP*, volume 2833 of *Lecture Notes in Computer Science*, 108–122. Springer.

Björk, M. 2011. Successful SAT Encoding Techniques. *J. Satisf. Boolean Model. Comput.*, 7(4): 189–201.

Bonassi, L.; De Giacomo, G.; Favorito, M.; Fuggitti, F.; Gerevini, A. E.; and Scala, E. 2023. Planning for Temporally Extended Goals in Pure-Past Linear Temporal Logic. In *ICAPS*, 61–69. AAAI Press.

Bonassi, L.; Gerevini, A. E.; and Scala, E. 2024. Dealing with Numeric and Metric Time Constraints in PDDL3 via Compilation to Numeric Planning. In *AAAI*, 20036–20043. AAAI Press.

Borgwardt, S.; Hoffmann, J.; Kovtunova, A.; and Steinmetz, M. 2021. Making DL-Lite Planning Practical. In *KR*, 641–645.

Bruttomesso, R.; Cimatti, A.; Franzén, A.; Griggio, A.; Hanna, Z.; Nadel, A.; Palti, A.; and Sebastiani, R. 2007. A Lazy and Layered SMT($\mathcal{BV}$) Solver for Hard Industrial Verification Problems. In *CAV*, volume 4590 of *Lecture Notes in Computer Science*, 547–560. Springer.

Cardellini, M.; Giunchiglia, E.; and Maratea, M. 2024. Symbolic Numeric Planning with Patterns. In *AAAI*, 20070–20077. AAAI Press.

Chen, D. Z.; and Thiébaux, S. 2024. Novelty Heuristics, Multi-Queue Search, and Portfolios for Numeric Planning. In *SOCS*, 203–207. AAAI Press.

Clarke, E. M.; Kroening, D.; and Yorav, K. 2003. Behavioral consistency of C and verilog programs using bounded model checking. In *DAC*, 368–371. ACM.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.*, 20: 61–124.

Gerevini, A. E.; Percassi, F.; and Scala, E. 2024. An Effective Polynomial Technique for Compiling Conditional Effects Away. In *AAAI*, 20104–20112. AAAI Press.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press.

Gigante, N.; and Scala, E. 2023. On the Compilability of Bounded Numeric Planning. In *IJCAI*, 5341–5349.

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; and Muise, C. 2019. *An Introduction to the Planning Domain Definition Language*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Helal, H.; and Lakemeyer, G. 2024. An Analysis of the Decidability and Complexity of Numeric Additive Planning. In *ICAPS*, 267–275. AAAI Press.

Helmert, M. 2002. Decidability and Undecidability Results for Planning with Numerical State Variables. In *AIPS*, 44–53. AAAI.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *J. Artif. Intell. Res.*, 20: 291–341.

Kiam, J. J.; Scala, E.; Jávega, M. R.; and Schulte, A. 2020. An AI-Based Planning Framework for HAPS in a Time-Varying Environment. In *ICAPS*, 412–420. AAAI Press.

Kouaiti, A. E.; Percassi, F.; Saetti, A.; McCluskey, T. L.; and Vallati, M. 2024. PDDL+ Models for Deployable yet Effective Traffic Signal Optimisation. In *ICAPS*, 168–177. AAAI Press.

Kroening, D.; and Strichman, O. 2016. *Decision procedures*. Springer.

Kuroiwa, R.; Shleyfman, A.; and Beck, J. C. 2022. LM-Cut Heuristics for Optimal Linear Numeric Planning. In *ICAPS*, 203–212. AAAI Press.

Kuroiwa, R.; Shleyfman, A.; and Beck, J. C. 2023a. Extracting and Exploiting Bounds of Numeric Variables for Optimal Linear Numeric Planning. In *ECAI*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, 1332–1339. IOS Press.

Kuroiwa, R.; Shleyfman, A.; and Beck, J. C. 2023b. NLM-CutPlan. In *IPC 2023*.

Kuroiwa, R.; Shleyfman, A.; Piacentini, C.; Castro, M. P.; and Beck, J. C. 2022. The LM-Cut Heuristic Family for Optimal Numeric Planning with Simple Conditions. *J. Artif. Intell. Res.*, 75: 1477–1548.

Leofante, F. 2023. OMTPlan: A Tool for Optimal Planning Modulo Theories. *J. Satisf. Boolean Model. Comput.*, 14(1): 17–23.

Mattmüller, R.; Geißer, F.; Wright, B.; and Nebel, B. 2018. On the Relationship Between State-Dependent Action Costs and Conditional Effects in Planning. In *AAAI*, 6237–6245. AAAI Press.

Micheli, A.; Bit-Monnot, A.; Röger, G.; Scala, E.; Valentini, A.; Framba, L.; Rovetta, A.; Trapasso, A.; Bonassi, L.; Gerevini, A. E.; Iocchi, L.; Ingrand, F.; Köckemann, U.; Patrizi, F.; Saetti, A.; Serina, I.; and Stock, S. 2025. Unified Planning: Modeling, manipulating and solving AI planning problems in Python. *SoftwareX*, 29: 102012.

Miller, C.; Gitina, K.; and Becker, B. 2011. Bounded Model Checking of Incomplete Real-time Systems Using Quantified SMT Formulas. In *MTV*, 22–27. IEEE Computer Society.

Nebel, B. 2000. On the Compilability and Expressive Power of Propositional Planning Formalisms. *J. Artif. Intell. Res.*, 12: 271–315.

Parkinson, S.; Longstaff, A.; Crampton, A.; and Gregory, P. 2012. The Application of Automated Planning to Machine Tool Calibration. In *ICAPS*. AAAI.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Artif. Intell. Res.*, 39: 127–177.

Scala, E.; Haslum, P.; Thiébaux, S.; and Ramírez, M. 2020. Subgoaling Techniques for Satisficing and Optimal Numeric Planning. *J. Artif. Intell. Res.*, 68: 691–752.

Shleyfman, A.; Gnad, D.; and Jonsson, P. 2023. Structurally Restricted Fragments of Numeric Planning–a Complexity Analysis. In *AAAI*, volume 37, 12112–12119.

Shleyfman, A.; Kuroiwa, R.; and Beck, J. C. 2023. Symmetry Detection and Breaking in Linear Cost-Optimal Numeric Planning. In *ICAPS*, 393–401. AAAI Press.

Speck, D.; Geißer, F.; Mattmüller, R.; and Torralba, Á. 2019. Symbolic Planning with Axioms. In *ICAPS*, 464–472. AAAI Press.

Taitler, A.; Alford, R.; Espasa, J.; Behnke, G.; Fiser, D.; Gimelfarb, M.; Pommerening, F.; Sanner, S.; Scala, E.; Schreiber, D.; Segovia-Aguas, J.; and Seipp, J. 2024. The 2023 International Planning Competition. *AI Mag.*, 45(2): 280–296.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *Artif. Intell.*, 168(1-2): 38–69.

Vallati, M.; Magazzeni, D.; Schutter, B. D.; Chrpa, L.; and McCluskey, T. L. 2016. Efficient Macroscopic Urban Traffic Models for Reducing Congestion: A PDDL+ Planning Approach. In *AAAI*, 3188–3194. AAAI Press.

Vizel, Y.; Nadel, A.; and Malik, S. 2017. Solving linear arithmetic with SAT-based model checking. In *FMCAD*, 47–54. IEEE.