On the Compilability of Bounded Numeric Planning

Nicola Gigante[1] , **Enrico Scala**[2]
[1]Free University of Bozen-Bolzano, Italy
[2]University of Brescia, Italy
nicola.gigante@unibz.it, enrico.scala@unibs.it

Abstract

Bounded numeric planning, where each numeric variable domain is bounded, is PSPACE-complete, but such a complexity result does not capture how hard it really is, since the same holds even for the practically much easier STRIPS fragment. A finer way to compare the difficulty of planning formalisms is through the notion of *compilability*, which has been however extensively studied only for classical planning by Nebel. This paper extends Nebel's framework to the setting of bounded numeric planning. First, we identify a variety of numeric fragments differing on the degree of the polynomials involved and the availability of features such as conditional effects and Boolean conditions; then we study the compilability of these fragments to each other and to the classical fragments. Surprisingly, numeric and classical planning with conditional effects and Boolean conditions can be compiled both ways preserving plan size exactly, while the same does not hold when targeting pure STRIPS. Our study reveals also that numeric fragments cluster into two equivalence classes separated by the availability of *incomplete initial state specifications*, a feature allowing to specify uncertainty in the initial state.

1 Introduction

Planning is the problem of finding a sequence of actions that, when applied to an initial state, reaches a state in which the goal is satisfied. Numeric planning is the variant encompassing both continuous and discrete state variables, and with action representations that can make use of linear and non-linear numeric expressions to both define applicability conditions, and state transitions. Over the past few years, there has been a renewed attention into planning problems involving numeric information—of which numeric planning represents a fundamental building block [Percassi *et al.*, 2021; Scala *et al.*, 2016]—both from an application standpoint [Kiam *et al.*, 2020; Vallati *et al.*, 2016; Bertolucci *et al.*, 2019; Parkinson *et al.*, 2012] and from the development of planners supporting its features [Piacentini *et al.*, 2018; Kuroiwa *et al.*, 2021; Scala *et al.*, 2020].

Unfortunately, Helmert [2002] showed that in the general case, and also in some more specific settings, numeric planning is undecidable. However, Helmert himself has shown as well the existence of some decidable fragments, obtained by restricting the kind of effects and conditions that can be expressed. In addition, he notes that one can also recover decidability, at least in the *discrete* case, with *bounded numeric planning*, *i.e.*, by imposing a bound on the value of the variables and looking only at those plans where the variables respect such bounds.

Once decidability questions are settled, one can turn their attention to the computational complexity of these problems. It turns out that bounded numeric planning is PSPACE-complete, as long as the kind of effects and conditions used are polynomial-space computable. Many other planning formalisms, from pure STRIPS [Bylander, 1994] to variants of temporal planning [Rintanen, 2007; Gigante *et al.*, 2020] and timeline-based planning [Bozzelli *et al.*, 2020; Della Monica *et al.*, 2020], have the same complexity but, for example, pure STRIPS is extremely easier to solve in practice. Hence, unfortunately, complexity theory tells us hardly anything about how hard these problems really are.

A way to compare planning formalisms at a finer granularity is the notion of *compilability*, that asks whether one formalism can be compiled into another. Compilations are different from complexity-theoretic reductions since they are tasked to translate the planning *domains* separately, in a way that allows later to cheaply plug different initial states and goal conditions.

Compilability has been studied extensively in the setting of classical planning by Nebel [2000], who set up an articulated framework for comparing the expressiveness of different fragments differing on the inclusion of features such as conditional effects, Boolean conditions, etc. An important feature of Nebel's framework is to account for the *growth of plan size* between a problem and its compiled version in a different formalism. In particular, a landmark result is that neither conditional effects nor Boolean conditions can be compiled away without a polynomial increase of plan size.

In this paper, we *extend* Nebel's framework to the setting of bounded numeric planning. We define a lattice of numeric planning languages that differ both on the degree of the polynomials allowed in numeric effects and conditions, and on the presence of the features already considered by Nebel. Then,

we compare these fragments to each other and to classical ones in terms of compilability, obtaining a complete picture. In particular, we show that:

1. numbers can be compiled away to classical planning if Boolean conditions and conditional effects are allowed;
2. conversely, Boolean conditions and conditional effects can be compiled away to numeric planning;
3. in numeric planning, conditional effects and Boolean conditions *can* be compiled away;

All the above compilations *preserve plan size exactly*. These results partition the numeric fragments into two equivalence classes that are distinguished by the presence of *partial initial state specifications*, *i.e.*, by the ability to partially specify the initial condition. To complete the picture, we prove that this separation is strict.

The presented results highlight a strict connection between numeric and conditional effects. The key of this connection appears to be the *state-dependency* of numeric and conditional effects, which allow both to perform complex computations as part of the application of a single action. This connection has not been precisely isolated before, and may pave the way for the cross-fertilization of the two fields, for instance in terms of search heuristics.

The paper is structured as follows. At first, Section 2 provides the required background about the considered planning problems. In Section 3 we recall Nebel's framework. Then, Section 4 provides the full picture of compilability between the considered numeric and classical fragments. Section 5 discusses the obtained results and possible future directions.

2 Planning Problems

Here we define the kind of planning problems considered.

Syntax and semantics Let $X = \{x, y, z, \dots\}$ be a set of *numeric* variables. A *numeric expression* e is an expression formed according to the following syntax:

$$e \coloneqq x \mid k \mid e \cdot e \mid e + e$$

where $x \in X$ is a variable and $k \in \mathbb{Z}$. Associativity and precedence rules hold as usual, and products $e_1 \cdot e_2$ are commonly shortened as $e_1 e_2$ when there is no ambiguity. For example, $(3xy + 4x)(2x + 4)$ is a numeric expression. Note that numeric expressions denote polynomial functions, although standard-form monomials such as x^2 have to be expressed as $x \cdot x$ because our syntax does *not* support exponentiation (see the complexity considerations for a motivation of this choice). The *degree* of a numeric expression is the maximum number of variables multiplied together when the expression is put into the standard sum-of-products form. For example, the above example expression is equivalent to $6xxy + 12xy + 8xx + 16x$, hence its degree is 3. A *numeric atom* is a formula ψ of the form $e \sim 0$ where e is a numeric expression and $\sim \in \{<, \leq, >, \geq, =\}$ is a relational operator.

Let $\Sigma = \{p, q, r, \dots\}$ be a set of *Boolean* variables, also called Boolean *atoms*. An *atom* is either a Boolean or numeric atom. A *literal* is either an atom ψ or its negation $\neg\psi$ (*e.g.*, $\neg p$ or $\neg(x + y = 0)$).

A *state specification* is a set of Boolean literals and numeric atoms of the form $x = k$ with $x \in X$ and $k \in \mathbb{Z}$. A state specification S is *complete* if any proposition $p \in \Sigma$ and any variable $x \in X$ are mentioned once in S. For example, if $\Sigma = \{p, q\}$ and $X = \{x, y\}$, then $S' = \{\neg p, x = 42\}$ is a state specification, and $S'' = \{\neg p, q, x = 42, y = 0\}$ is a complete one.

A *condition* is a formula ϕ observing the following syntax:

$$\phi \coloneqq \top \mid \bot \mid \ell \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi$$

where ℓ is a literal.

An *elementary effect* is either a Boolean literal or an assignment of the form $x \coloneqq e$ or $x \coloneqq x + e$ where $x \in X$ and e is a numeric expression. An *effect* is a pair (Γ, E), denoted as $\Gamma \Rightarrow E$, where Γ is a set of conditions and E is a set of elementary effects. An effect is *conditional* or *unconditional* depending on whether $\Gamma \neq \varnothing$ or $\Gamma = \varnothing$, respectively.

An *action* is a pair $a = \langle \mathrm{pre}(a), \mathrm{eff}(a) \rangle$ where $\mathrm{pre}(a)$ is a set of conditions and $\mathrm{eff}(a)$ is a set of effects. A *bound function* for X is a function $\beta : X \to \mathbb{Z}^2$ that maps each variable $x \in X$ to a pair $\beta(x) = (l, u)$ such that $l \leq u$.

We are now ready to define our planning problems.

Definition 1 (Planning problem)**.** *A* planning domain *is a tuple $\mathcal{D} = \langle \Sigma, X, \beta, A \rangle$ where Σ is a set of Boolean variables, X is a set of numeric variables, β is a bound function for X, and A is a set of actions.*

A planning problem *(or* instance*) is a tuple $\mathcal{P} = \langle \mathcal{D}, I, G \rangle$ where I is a state specification and G is a set of conditions.*

We can now briefly define the semantics of planning problems. A *state* is a pair $s = (\sigma, \nu)$ where $\sigma : \Sigma \to \{\top, \bot\}$ and $\nu : X \to \mathbb{Z}$ are functions that give (Boolean or numeric) values to the (Boolean or numeric) variables of the domain. The semantics of a *condition* ϕ being satisfied by a state s, written $s \models \phi$, is defined as one would expect. A set of conditions Γ is satisfied on s, denoted $s \models \Gamma$, if all the conditions in Γ are satisfied. A state $s = (\sigma, \nu)$ is *eligible* if for all $x \in X$, if $\beta(x) = (l, u)$, then $l \leq \nu(x) \leq u$. A state specification S induces a set of states $[\![S]\!] = \{s \mid S \models s\}$. It is called *consistent* if $|[\![S]\!]| > 0$. Note that if S is a *complete* specification, it induces at most a single state, *i.e.*, $|[\![S]\!]| \leq 1$. An effect $\Gamma \Rightarrow E$ is *enabled* on s if $s \models \Gamma$. Two effects $\Gamma_1 \Rightarrow E_1$ and $\Gamma_2 \Rightarrow E_2$ are *conflicting* on s if they are both enabled on s and E_1 and E_2 contain complementary literals (*i.e.*, p and $\neg p$) or they contain two numeric effects $x \coloneqq e_1$ and $x \coloneqq e_2$ that would assign different values to the same variable x when evaluated on s. An action a is applicable to a state s if $s \models \mathrm{pre}(a)$ and none of its effects are conflicting on s. If a is applicable to s, $a(s)$ is the state obtained from s by applying all the enabled effects of a (the semantics of the application of effects is defined as one would expect, and we leave it out for space concerns).

A *plan* is a finite sequence of actions $\pi = \langle a_1, \dots, a_n \rangle$. Given an initial state s_0, a plan $\pi = \langle a_1, \dots, a_n \rangle$ induces a sequence of states $\pi(s_0) = \langle s_0, \dots, s_n \rangle$ where $s_{i+1} = a_{i+1}(s_i)$ for all $i \geq 0$. This definition is well-behaved and $\pi(s_0)$ exists only if a_{i+1} is applicable to s_i for all $i \geq 0$.

Definition 2. *Given a planning problem $\mathcal{P} = \langle \mathcal{D}, I, G \rangle$, a plan π is a* solution plan *for $\mathcal{P}$ if I is consistent and, for all*

eligible *states* $s_0 \in [\![I]\!]$, $\pi(s_0)$ *exists, its last state* $s_n \models G$, *and all the states of* $\pi(s_0)$ *are eligible.*

Fragments The problems with complete state specifications as for Definition 1 can be expressed by PDDL2.1 [Fox and Long, 2003; Haslum *et al.*, 2019]. Here, we focus on the relationships between the expressive power of some restricted fragments. Following Nebel [2000], we introduce a compact notation to identify the fragments obtained by restricting different aspects of the language.

Let $\mathcal{D} = \langle \Sigma, X, \beta, A \rangle$ be a domain and $\mathcal{P} = \langle \mathcal{D}, I, G \rangle$ be a problem. At first, note that if $X = \varnothing$, $\mathcal{P}$ is a *classical* planning problem, with no numeric features, of the kind already studied by Nebel. Hence, we use the symbols N (for *numeric*) and S (for *STRIPS*) to denote the language of planning problems where $X \neq \varnothing$ and $X = \varnothing$, respectively. Then, we use additional symbols as sub/superscripts to denote the availability of certain features. The following are used by Nebel:

1. I: the initial state specification can be *incomplete*;
2. C: effects are allowed to be *conditional*;
3. L: conditions are allowed to be *literals*;
4. B: the general Boolean syntax for conditions is allowed.

We introduce further notation for numeric features:

5. P^n: numeric expressions are allowed up to degree n;
6. P: numeric expressions of any degree are allowed.

Thus we denote as S_B and $\mathsf{N}_B^{C,E}$ with $B \subseteq \{\mathsf{I}, \mathsf{C}, \mathsf{L}, \mathsf{B}\}$ and $C, E \subseteq \{\mathsf{P}^n, \mathsf{P}\}$ the language of planning problems (respectively with $X = \varnothing$ and $X \neq \varnothing$), where the features in B are allowed, the features in C are allowed in conditions, and the features in E are allowed in effects. It is worth to note that B subsumes L and P subsumes P^n. Note that features L and B refer to conditions appearing both as action's preconditions, in conditional effects, and as goals.[1]

For example, $\mathsf{N}_{\mathsf{BCI}}^{\mathsf{P},\mathsf{P}}$ is the most general language defined in Definition 1, while $\mathsf{S}_{\mathsf{BCI}}$ is the most general *classical* language that we consider (and that has been considered by Nebel as well). Instead, $\mathsf{N}^{\mathsf{P}^1,\mathsf{P}^0}$ is the language of numeric planning problems where the underlying classical language is the most restrictive, numeric expressions in conditions are *linear*, and numeric effects are limited to assignments and increase/decrease by a constant.

Complexity considerations It is useful to recall some computational complexity aspects of the problems studied here. It is known that the plan existence problem for $\mathsf{S}_{\mathsf{BCI}}$ is PSPACE-complete [Bylander, 1994; Nebel, 2000]. Instead, numeric planning is in general *undecidable* [Helmert, 2002]. However, Helmert also notes that for integer variables, imposing bounds recover decidability, and it is folklore knowledge that numeric planning is PSPACE-complete with any kind of numeric effects and conditions, as long as they are polynomial-space computable.

This assumption includes numeric expressions as defined above, hence the plan existence problem for $\mathsf{N}_{\mathsf{BC}}^{\mathsf{P},\mathsf{P}}$ is PSPACE-complete as well. Instead, the assumption does not hold if succinct exponentiation is allowed, such as e^k for some expression e and $k \in \mathbb{N}$, because one needs an exponential amount of space to represent the result of e^k. For this reason, it is likely that numeric planning with general exponentiation does not indeed belong to PSPACE. Also considering that PDDL 2.1 does not support it as well, this is not a too restrictive assumption.

With *incomplete state specifications*, the situation is a bit more involved. The semantics in this case involves a universal quantification over the eligible states that satisfy the initial state specification, so it is not obvious *a priori* whether an efficient nondeterministic algorithm could be given. Indeed, this setting is similar to *conformant planning* [Haslum and Jonsson, 1999], which in its usual formulation is EXPSPACE-complete. However, Nebel [2000] shows that $\mathsf{S}_{\mathsf{BCI}}$ is still PSPACE-complete, and this holds because the set of initial states in conformant planning is usually defined as a Boolean condition, while our state specifications are only partial assignments. Nebel's argument does not hold in the numeric case, but we will show, as a consequence of our compilations, that $\mathsf{N}_{\mathsf{BCI}}^{\mathsf{P},\mathsf{P}}$ is PSPACE-complete as well.

3 Compilation Schemes

We are concerned with the *compilability* of the fragments defined above between themselves. Since all those formalisms, from S to $\mathsf{N}_{\mathsf{BCI}}^{\mathsf{P},\mathsf{P}}$, are PSPACE-complete, we know there are polynomial-time *reductions* between the respective decision problems. However, when talking about *compilations*, similarly to Nebel [2000], we look for something stronger, *i.e.*, transformations from/to planning *domains*, where initial and goal conditions can be plugged subsequently after a further but very light translation. Here we slightly extend the definition by Nebel [2000] to account for numeric features:

Definition 3 (Compilation scheme)**.** *A* compilation scheme *is a tuple of functions* $f = \langle f_\xi, f_i, f_g, t_i, t_g \rangle$ *that induce a function from instances* $\mathcal{P} = \langle \mathcal{D}, I, G \rangle$*, with* $\mathcal{D} = \langle \Sigma, X, \beta, A \rangle$*, to instances* $f(\mathcal{P})$*, as follows:*

$$f(\mathcal{P}) = \langle f_\xi(\mathcal{D}), f_i(\mathcal{D}) \cup t_i(\Sigma, X, I), f_g(\mathcal{D}) \cup t_g(\Sigma, X, G) \rangle$$

where:

1. *there is a solution plan for* $\mathcal{P}$ *iff there is one for* $f(\mathcal{P})$*;*
2. *the functions* t_i *and* t_g *are polynomial-time computable and are* modular*,* i.e.*, if* $\Sigma = \Sigma_1 \cup \Sigma_2$*, and* S *is a state specification, then:*
$$t_x(\Sigma, X, S) = t_x(\Sigma_1, X, S|_{\Sigma_1}) \cup t_x(\Sigma_2, X, S|_{\Sigma_2})$$
3. *the size of* $f(\mathcal{P})$ *is* polynomial *in the size of* $\mathcal{P}$*.*

Intuitively, Definition 3 states that we can compile the planning domain separately once (with f_ξ), and then, when given a complete instance, form the compiled instance by forming the initial and goal conditions with some components that depend *only* on the domain (with f_i and f_g) and some components that depend *only* on the original initial and goal conditions (with t_i and t_g). These have to be easy and simple mappings, *i.e.*, the "hard work" has to be done by the mapping between the domains.

[1] Nebel [2000] considers goals to always be sets of literals, but states that his results apply as well on problems with condition goals.

When studying compilations, we are also interested in the length of the plans of the compiled problems. Again, following Nebel [2000], we say that a language L can be compiled to L' *preserving plan size exactly*, *linearly*, or *polynomially*, written $L \preceq^1 L'$, $L \preceq^c L'$, or $L \preceq^p L'$, respectively, if there is a compilation scheme f such that, for any problem $\mathcal{P}$ of L and any solution plan π for $\mathcal{P}$, then $f(\mathcal{P})$ has a solution plan π' such that $|\pi'| \leq |\pi| + k$, $|\pi'| \leq c|\pi| + k$, and $|\pi'| \leq p(|\pi|, |\mathcal{P}|)$, respectively, for some $k, c \in \mathsf{N}$ and some polynomial p. If $L \preceq^x L'$ and $L' \preceq^x L$, with $x \in \{1, c, p\}$, we write $L \approx^x L'$.

This definition of compilability is quite more restrictive than the complexity-theoretic reducibility relation. For example, we recall some results proved by Nebel [2000],

Proposition 1. *The following holds:*

1. $\mathsf{S_C} \not\preceq^c \mathsf{S_{BI}}$, $\mathsf{S_B} \not\preceq^c \mathsf{S_{LCI}}$, *and* $\mathsf{S_{BI}} \not\preceq^p \mathsf{S_{BC}}$.
2. $\mathsf{S_{LCI}} \preceq^p \mathsf{S}$ *and* $\mathsf{S_{BC}} \preceq^p \mathsf{S}$

In other words, while conditional effects and Boolean conditions can be compiled away preserving plan size polynomially, this cannot be done preserving plan size linearly, and when Boolean conditions are paired with incomplete state specifications, plan size cannot even be preserved polynomially. With compilability, we obtain a much more fine-grained lens to tell how hard a planning formalism is, w.r.t. complexity theory, which puts everything into the PSPACE pot.

4 Compilability Results

Now that the framework of compilability has been set up, we can provide some results. Given that the compilability between classical formalisms has been already studied exhaustively by Nebel [2000], the most natural question is how numerical fragments fit in the picture. To answer this question, we provide results that classify the numerical fragments defined in Section 2 in terms of compilability between themselves and torwards classical fragments. An overview of the results is pictured in Figure 1.

We first show an initial result about the relationship of numerical fragments with different allowed polynomial degree. It turns out the complexity of the conditions can be transferred to the effects if needed, preserving plan size exactly.

Theorem 1. $\mathsf{N}_{\mathsf{BCI}}^{\mathsf{P}^n,\mathsf{P}^m} \preceq^1 \mathsf{N}_{\mathsf{BCI}}^{\mathsf{P}^1,\mathsf{P}^d}$ *where* $d = \max(m, n)$.

Proof. This first compilation scheme is based on the idea that numeric expressions $e \sim 0$ used in the conditions of actions or conditional effects can be replaced by simple comparisons $x_e \sim 0$ where x_e is an additional numeric variable that is always kept to hold the value of e.[2]

Given two numeric expressions e_1 and e_2, let us denote as $e_1[x/e_2]$ the expression obtained by replacing with e_2 any occurrence of x in e_1. Then, to compute the correct bounds for the computation of an expression e, we define $\beta(e)$ as:

$$\beta(k) = (k, k)$$

[2]The procedure subsumes a result by Kuroiwa *et al.* [2021] who have shown that linear numeric conditions can be compiled into so called restricted numeric tasks [Hoffmann, 2003], *i.e.*, into bound conditions of the form $x \geq k$.

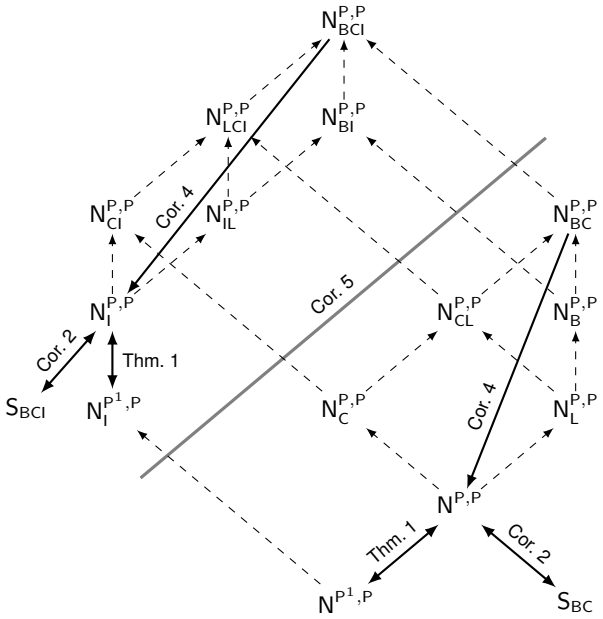

Figure 1: Compilability landscape of numerical and classical fragments. Dashed arrows are natural inclusions between fragments, solid arrows represent compilability preserving plan size exactly. The gray line separates the two equivalence classes modulo $\approx^1$, whose separation is strict.

$$\beta(e_1 + e_2) = (l_1 + l_2, u_1 + u_2)$$
$$\beta(e_1 \cdot e_2) = (l_1 \cdot l_2, u_1 \cdot u_2)$$

where $\beta(e_1) = (l_1, u_1)$ and $\beta(e_2) = (l_2, u_2)$.

Now, let $\mathcal{D} = \langle \Sigma, X, \beta, A \rangle$. The core component f_ξ of the compilation scheme is $f_\xi(\mathcal{D}) = \langle \Sigma, X', \beta', A' \rangle$ where:

$$X' = X \cup \{x_e \mid e \text{ appears in a condition } e \sim 0 \text{ in } \mathcal{D}\}$$

$$\beta'(x) = \begin{cases} \beta(x) & \text{if } x \in X \\ \beta(e) & \text{if } x = x_e \text{ for some } e \end{cases}$$

and A' is defined as follows. For each $a \in A$ and any condition $e \sim 0$ appearing in $\mathrm{pre}(a)$ or in conditional effects in $\mathrm{eff}(a)$, we replace the condition with $x_e \sim 0$. But now, we have to ensure that at any point in the plan, the value of x_e reflects the evaluation of e at that point. Hence for any other $b \in A$, let $\Gamma \Rightarrow E$ be an effect of b. Let $x_1, \ldots, x_n$ be the variables that are assigned in E (to expressions $e_1, \ldots, e_n$) and are mentioned by e. Then, we add an effect $\Gamma \Rightarrow E'$ where $E' = \{x_e := e[x_i/e_i]\}$.

Then, we have to take care of the initial value of x_e. If I is a complete state specification, we can get from I the values $v_1, \ldots, v_n$ of the variables $x_1, \ldots, x_n$ mentioned by e, compute its value v_e, and set $f_i(\mathcal{D}) = \bigcup_e \{x_e = v_e\}$. If I is incomplete, we set $f_i(\mathcal{D}) = \bigcup_e \{x_e = e\}$.

To translate the goal condition, the function f_g replaces any numeric atom $e \sim 0$ with $x_e \sim 0$. The functions t_i and t_g do nothing, *i.e.*, $t_i(\Sigma, X, I) = t_g(\Sigma, X, G) = \varnothing$.

It can be checked that $f = \langle f_\xi, f_i, f_g, t_i, t_g \rangle$ so defined satisfies Definition 3. Note that f preserves plan size exactly since actions are mapped 1-to-1 in the compiled problem.

Moreover, if the degree of the expressions used in conditions is higher than those used in effects, f increases the degree of effects accordingly. Hence we go from $\mathsf{N}^{\mathsf{P}^n,\mathsf{P}^m}_{\mathsf{BCI}}$ to $\mathsf{N}^{\mathsf{P}^1,\mathsf{P}^d}_{\mathsf{BCI}}$ where $d = \max(m, n)$. □

Now we can turn to the relationship between numeric and classical planning. We first show that numbers alone are powerful enough to compile away both conditional effects and Boolean conditions, preserving plan size exactly.

Theorem 2. $\mathsf{S}_{\mathsf{BCI}} \preceq^1 \mathsf{N}^{\mathsf{P},\mathsf{P}}_{\mathsf{I}}$ *and* $\mathsf{S}_{\mathsf{BC}} \preceq^1 \mathsf{N}^{\mathsf{P},\mathsf{P}}$.

Proof. To define this compilation scheme we recall that Boolean functions, such as those described by Boolean conditions of our planning problems, can be represented by particular real polynomials (see *e.g.*, [O'Donnell, 2014]). The basic underlying observation is that arithmetic multiplication between numeric values ranging in $\{0, 1\}$ exactly corresponds to Boolean conjunction.

In detail, given a set Σ of Boolean variables, we define a set of numeric variables as $X_\Sigma = \{x_p \mid p \in \Sigma\}$. Then, given a Boolean condition ϕ, we define the numeric expression $\mathrm{num}(\phi)$ as follows:

$$\begin{aligned}
\mathrm{num}(\top) &= 1\\
\mathrm{num}(p) &= x_p\\
\mathrm{num}(\neg\phi) &= 1 - \mathrm{num}(\phi)\\
\mathrm{num}(\phi_1 \wedge \phi_2) &= \mathrm{num}(\phi_1) \cdot \mathrm{num}(\phi_2)\\
\mathrm{num}(\phi_1 \vee \phi_2) &= \mathrm{num}(\neg(\neg\phi_1 \wedge \neg\phi_2)) =\\
&= 1 - ((1 - \mathrm{num}(\phi_1)) \cdot (1 - \mathrm{num}(\phi_2)))
\end{aligned}$$

Note that one can prove by a simple induction that if $x_p \in \{0, 1\}$ for $x_p \in X_\Sigma$, then the value of $\mathrm{num}(\phi)$ is either 0 or 1, and it corresponds to the truth value of ϕ.

Now, let $\mathcal{D} = \langle \Sigma, \varnothing, \varnothing, A\rangle$ be a planning domain in the $\mathsf{S}_{\mathsf{BCI}}$ language. We define $f_\xi(\mathcal{D}) = \langle \varnothing, X_\Sigma, \beta, A'\rangle$ where $\beta(x) = (0, 1)$ for all $x \in X_\Sigma$, and A' has exactly one action $a' = \langle \mathrm{pre}(a'), \mathrm{eff}(a')\rangle$ for each $a \in A$, defined as follows. First of all, $\mathrm{pre}(a') = (\mathrm{num}(\mathrm{pre}(a)) = 1)$. Then, for each $p \in \Sigma$, let Δ^+_p and Δ^-_p be the following:

$$\Delta^+_p = \{\textstyle\bigwedge_{\phi\in\Gamma} \phi \mid (\Gamma \Rightarrow E) \in \mathrm{eff}(a) \text{ and } p \in E\}$$
$$\Delta^-_p = \{\textstyle\bigwedge_{\phi\in\Gamma} \phi \mid (\Gamma \Rightarrow E) \in \mathrm{eff}(a) \text{ and } \neg p \in E\}$$

That is, Δ^+_p and Δ^-_p collect all the sufficient conditions for p to become true or false, respectively, when applying a. Now, we can define:

$$\phi_p \equiv (p \wedge \textstyle\bigwedge_{\phi\in\Delta^-_p} \neg\phi) \vee (\bigvee_{\phi\in\Delta^+_p} \phi)$$
$$\mathrm{eff}(a') = \{\varnothing \Rightarrow \{x_p \coloneqq \mathrm{num}(\phi_p)\} \mid p \in \Sigma\}$$

The rest of the compilation scheme is easy to define:

$$\begin{aligned}
f_i(\mathcal{D}) &= \varnothing\\
f_g(\mathcal{D}) &= \varnothing\\
t_i(\Sigma, X, I) &= \{x_p = 1 \mid p \in I\} \cup \{x_p = 0 \mid \neg p \in I\}\\
t_g(\Sigma, X, G) &= \{num(\phi) = 1 \mid \phi \in G\}
\end{aligned}$$

It can be checked that $f = \langle f_\xi, f_i, f_g, t_i, t_g\rangle$ so defined is a compilation scheme, by Definition 3, from $\mathsf{S}_{\mathsf{BCI}}$ to $\mathsf{N}^{\mathsf{P},\mathsf{P}}_{\mathsf{I}}$, and Boolean conditions and conditional effects become unnecessary in the compiled problem. Moreover, given any $\mathcal{P}$ of $\mathsf{S}_{\mathsf{BCI}}$, $f(\mathcal{P})$ has exactly one action for each action of $\mathcal{P}$, and this reflects on the size of the plans which are preserved exactly. Finally, note that f does not introduce incomplete state specifications if not present in the original problem. Hence, when starting from a problem in S_{BC}, the resulting problem is in $\mathsf{N}^{\mathsf{P},\mathsf{P}}$. □

The result of Theorem 2 could actually be sharpened. When building $\mathrm{num}(\phi)$, note that if $x_p \in \{0, 1\}$, $x_p \cdot x_p = x_p$ (which is nothing more than the fact that $p \wedge p \equiv p$). Hence, if we put $\mathrm{num}(\phi)$ in standard form, each variable would appear at most with degree 1, so $\mathrm{num}(\phi)$ has at most degree n where n is the number of variables mentioned in ϕ. This means that from a given planning problem $\mathcal{P}$ in S_{BC} (resp. in $\mathsf{S}_{\mathsf{BCI}}$) we obtain a planning problem in $\mathsf{N}^{\mathsf{P}^n,\mathsf{P}^m}$ (resp. in $\mathsf{N}^{\mathsf{P}^n,\mathsf{P}^m}_{\mathsf{I}}$), where n and m are the maximum number of Boolean variables mentioned in the precondition of a single action or in the condition of a single conditional effect, respectively.

Theorem 2 and Proposition 1 imply that numbers cannot be compiled away, preserving plan size linearly, without making use of conditional effects and Boolean conditions.

Corollary 1. $\mathsf{N}^{\mathsf{P},\mathsf{P}}_{\mathsf{I}} \not\preceq^c \mathsf{S}_{\mathsf{I}}$ *and* $\mathsf{N}^{\mathsf{P},\mathsf{P}} \not\preceq^c \mathsf{S}$.

Now we can show in some sense the converse of Theorem 2, that is, that numeric planning can be compiled to classical planning preserving plan size exactly if we allow conditional effects and Boolean conditions.

This result requires some background about the complexity of *Boolean circuits* and *arithmetic circuits* that we are going to briefly cover here. Boolean circuits are a common computation model for Boolean functions, *i.e.*, functions $f : \{0, 1\}^n \to \{0, 1\}^m$, that has been extensively studied in the computational complexity literature for decades. A Boolean circuit is a directed acyclic graph made of logic gates (in our case, and, or, and not gates) and input variables, with edges going from input variables or gates outputs to gates inputs. Some of the gates are marked as the *output* gates of the circuit. The *depth* of a circuit is the maximum length of a path from an input variable to an output node. A Boolean *formula* is a circuit where the output of each gate is used at most once (we say the *fan-out* is 1). The study of the minimal size and depth of circuits that compute given functions is an interesting and complex field. For more details we refer the reader to the many textbooks on the topic (see *e.g.*, Wegener; Vollmer [1987; 1999]).

Another well-studied computation model that will come useful here is the *arithmetic circuits*, which are similar to Boolean ones with the difference that the gates compute arithmetic operations (in our case, sums and products) and the inputs are numbers (or elements in a field, more generally). Similar to Boolean formulas, arithmetic formulas are arithmetic circuits with fan-out 1. Note that numeric expressions defined in Section 2 can be directly seen as arithmetic formulas. For more details the reader can refer to [Vollmer, 1999].

Here, we are interested in how to compute arithmetic formulas with Boolean formulas. To answer this question, we need a small detour through circuit complexity results. In all the following statements, it is understood that the circuits and formulas whose existence is stated can be obtained in polynomial time. To start, the following result links evaluation of arithmetic formulas to arithmetic circuits.

Proposition 2 (Buss *et al.* [1992])**.** *Arithmetic formulas of size n can be computed over semi-rings by arithmetic circuits of size $\mathcal{O}(n^k)$, for some $k > 0$, and depth $\mathcal{O}(\log n)$.*

Note that integers modulo m, which is what concerns us, form a semi-ring and thus apply to Proposition 2.

Then, the following result, proved by Jung [1985] and also remarked by von zur Gathen and Seroussi [1991], connects arithmetic circuits over the integers to Boolean circuits.

Proposition 3. *Any arithmetic circuit of size s with n inputs and depth d can be computed over the* integers modulo m *by a Boolean circuit of size $\mathcal{O}(s^k)$ and depth $\mathcal{O}(d \log^* n)$.*

Here, $\log^* n$ is the *iterated logarithm function*, which grows very slowly. In particular, $\log^* m \in \mathcal{O}(\log m)$.

Finally, the size of Boolean formulas is strictly related to the size of Boolean circuits. In particular, it is well known that formulas grow exponentially on the depth of circuits.

Proposition 4. *Any Boolean circuit of size n and depth d is equivalent to a Boolean formula of size $\mathcal{O}(n2^d)$.*

The above results have the following simple consequence.

Proposition 5. *A numeric expression of size m can be computed by a Boolean formula of size polynomial in m.*

Proof. Consider a numeric expression e of size m involving n variables. We can see it as an arithmetic formula and apply Proposition 2 to get an arithmetic circuit of size $m^{\mathcal{O}(1)}$ and depth $d \in \mathcal{O}(\log m)$. Then, Proposition 3 finds a Boolean circuit of size still $m^{\mathcal{O}(1)}$ and depth $\mathcal{O}(d \log^* n) = \mathcal{O}(\log m \log^* n)$. Since $n = \mathcal{O}(m)$, the depth is $\mathcal{O}(\log^2 m)$. Then, by Proposition 4 we can get an equivalent Boolean formula of size $\mathcal{O}(m^{O(1)} 2^{\log m}) = m^{\mathcal{O}(1)}$. □

Recall that a Boolean formula in this setting is a circuit with fan-out 1, not some string of symbols such as the Boolean conditions of Section 2, but the connection is strict. From a Boolean formula with n outputs we can immediately recover n single-output Boolean formulas, and then n Boolean conditions usable to compute the single output bits.

Thanks to the above results, we can simulate the arithmetic operations expressed in numeric actions by means of Boolean conditions used to define suitable conditional effects.

Theorem 3. $\mathsf{N}^{\mathsf{P},\mathsf{P}}_{\mathsf{BC}} \preceq^1 \mathsf{S}_{\mathsf{BC}}$ *and* $\mathsf{N}^{\mathsf{P},\mathsf{P}}_{\mathsf{BCI}} \preceq^1 \mathsf{S}_{\mathsf{BCI}}$

Proof. Let $\mathcal{P} = \langle \mathcal{D}, I, G \rangle$ and $\mathcal{D} = \langle \Sigma, X, \beta, A \rangle$ be a problem and a domain from $\mathsf{N}^{\mathsf{P},\mathsf{P}}_{\mathsf{BC}}$. We start by simplifying the problem in two ways. First, we can normalize the bounds given by β in the following sense. We can assume *w.l.o.g.* that for each $x \in X$ there is a $w_x > 0$ such that $\beta(x) = (-2^{w_x}, 2^{w_x} - 1)$. If this is not the case, and $\beta(x) = (l, u)$, we can find a suitable w_x such that $-2^{w_x} \leq l$ and $2^{w_x} - 1 \geq u$, set $\beta(x)$ accordingly, and add to any action of the domain two preconditions $x \geq l$ and $x \leq u$. Then, thanks to Theorem 1, we can assume *w.l.o.g.* that $\mathcal{P}$ is of $\mathsf{N}^{\mathsf{P}^1,\mathsf{P}}_{\mathsf{BC}}$, *i.e.*, conditions are expressed with numeric atoms of degree 1. Note that, in particular, we can assume *w.l.o.g.* that all such atoms are of the form $x \geq 0$, with $x \in X$.

With these assumptions in place, we can simulate the arithmetics of the numeric planning problem by *bit-blasting*, *i.e.*, representing each numeric variable x with a set of $w_x + 1$ Boolean variables $\bar{x} = \{x_0, \ldots, x_{w_x}\}$. By using a *two's complement* representation, the most significant bit x_{w_x} also acts as a *sign bit*, *i.e.*, we will have $w_x = 0$ if $x \geq 0$ and $w_x = 1$ otherwise. Recall that a $(w+1)$-bits two's complement number represents an integer between -2^w and $2^w - 1$, which explains the assumptions on bounds made above. Recall as well that arithmetics in two's complement coincides with arithmetics in integers *modulo* 2^{w+1}. For each numeric variable x we also keep an additional Boolean variable o^x that will be used to keep track of overflows.

Now, the component f_ξ of the compilation scheme can be defined such that $f_\xi(\mathcal{D}) = \langle \Sigma', \varnothing, \varnothing, A' \rangle$, where $\Sigma' = \Sigma \cup \{x_0, \ldots, x_{w_x}, o^x \mid x \in X\}$, and A' is defined as follows. Each $a \in A$ is replaced by an action $a' = \langle \mathrm{pre}(a), \mathrm{eff}(a) \rangle$. Let us see at first how to deal with the precondition. Intuitively, to encode $x \geq 0$ we only need to test the sign bit of x. Given a condition ϕ, let $\mathrm{enc}(\phi)$ be the following:

$$\begin{aligned} \mathrm{enc}(p) &= p \quad \text{for } p \in \Sigma \\ \mathrm{enc}(x \geq 0) &= \neg x_{w_x} \quad \text{for } x \in X \\ \mathrm{enc}(\neg \phi_1) &= \neg\, \mathrm{enc}(\phi_1) \\ \mathrm{enc}(\phi_1 \vee \phi_2) &= \mathrm{enc}(\phi_1) \vee \mathrm{enc}(\phi_2) \end{aligned}$$

Then, we can define $\mathrm{pre}(a') = \mathrm{enc}(\mathrm{pre}(a))$. The conditional effects are the hardest part. Let e be a numeric expression. We can compute the number of bits needed to represent any possible value held by e, given the bounds of the variables involved in the expression, as $w_e + 1$, where w_e is defined as:

$$\begin{aligned} w_k &= \lceil \log(|k|) \rceil + 1 \quad \text{for } k \in \mathbb{Z} \\ w_{e_1 + e_2} &= \max(w_{e_1}, w_{e_2}) + 1 \\ w_{e_1 \cdot e_2} &= w_{e_1} + w_{e_2} + 1 \end{aligned}$$

For each numeric expression e, thanks to Proposition 5, we can obtain a set of Boolean conditions $\phi_0, \ldots, \phi_{w_e}$ computing the bits of the value of e given the bits of the involved variables. This computation is done *modulo* 2^{w_e+1}, which corresponds exactly to two's complement arithmetic over (w_e+1)-bits words. If involved variables have a lower bit width, they can be trivially extended to the higher bit width.

Now, let us assume *w.l.o.g.* that each effect $\Gamma \Rightarrow E$ in $\mathrm{eff}(a)$ is such that $|E| = 1$; when $|E| > 1$ we can split $\Gamma \Rightarrow E$ in $|E|$ different effects each with the same condition Γ. Now, let Γ be a set of conditions. We can define $\mathrm{enc}(\Gamma) = \{\mathrm{enc}(\psi) \mid \psi \in \Gamma\}$, similarly to the action's precondition. Then, let $\Gamma \Rightarrow E$ be an effect. If E contains a Boolean effect, we can define $\mathrm{enc}(\Gamma \Rightarrow E) = \mathrm{enc}(\Gamma) \Rightarrow E$.

Otherwise, if $E = \{x \coloneqq e\}$, the effect can be encoded as:

$$\mathrm{enc}(\Gamma \Rightarrow E) = \bigcup_{i=0}^{w_x} \left\{ \begin{array}{l} \Gamma_i^+ \Rightarrow \{x_i\}, \\ \Gamma_i^- \Rightarrow \{\neg x_i\}, \\ \Gamma^o \Rightarrow \{o^x\} \end{array} \right\}$$

where:

$$\Gamma_i^+ = \mathrm{enc}(\Gamma) \cup \{\phi_i\}$$
$$\Gamma_i^- = \mathrm{enc}(\Gamma) \cup \{\neg\phi_i\}$$
$$\Gamma^o = \mathrm{enc}(\Gamma) \cup \{\neg\big(\bigwedge_{i=w_x}^{w_e} \phi_i \vee \bigwedge_{i=w_x}^{w_e} \neg\phi_i\big)\}$$

Then, we can define $\mathrm{eff}(a')$ as the union of $\mathrm{enc}(\Gamma \Rightarrow E)$ for all the effects $(\Gamma \Rightarrow E) \in \mathrm{eff}(a)$. Intuitively, what we do here is to set each bit of the target variable with different conditional effects that depend on the value of the Boolean conditions ϕ_i representing the i-th bit of the numeric expression to be assigned to the numeric variable. We detect overflow when the value of e exceeds the value of x with an additional fluent. The extra bits of e can be safely truncated if they are equal to the first non-truncated bit. For example, the 8-bit word 11110110 (-10) can be assigned to a 5 bit word since the four most significant bits are equal (10110 is still -10).

Now, given $x_i \in \Sigma'$ and $k \in \mathbb{Z}$, let $x_i(k)$ be x_i if the i-th bit of the $(w_x + 1)$-bits two's complement representation of k is 1, and $\neg x_i$ otherwise. We have:

$$f_i(\mathcal{D}) = \{\neg x^o \mid x \in X\}$$
$$f_g(\mathcal{D}) = \{\neg x^o \mid x \in X\}$$
$$t_i(\Sigma, X, I) = I \cup \{x_i(k) \mid (x = k) \in I, i \in \{0, \ldots, w_x\}\}$$
$$t_g(\Sigma, X, G) = \{\ell \mid \ell \in G \text{ is Boolean}\} \cup \{\mathrm{enc}(\phi) \mid \phi \in G\}$$

It can be checked that $f = \langle f_\xi, f_i, f_g, t_i, t_g\rangle$ so defined is a compilation scheme by Definition 3. Moreover, it is easy to see that the initial state specification remains *complete*, if it is so before the compilation. Finally, the plan size is preserved exactly as each action of the original problem is replaced 1-to-1 with the new actions. □

Theorems 2 and 3, together, tell us that $\mathsf{N}^{\mathsf{P},\mathsf{P}}$ and S_{BC} are strongly related as one can be compiled in the other preserving plan size exactly.

Corollary 2. $\mathsf{S}_{\mathsf{BC}} \approx^1 \mathsf{N}^{\mathsf{P},\mathsf{P}}$ *and* $\mathsf{S}_{\mathsf{BCI}} \approx^1 \mathsf{N}_{\mathsf{I}}^{\mathsf{P},\mathsf{P}}$

As promised in Section 2, Theorem 3 also gives us a complexity bound on the plan existence problem for $\mathsf{N}_{\mathsf{BCI}}^{\mathsf{P},\mathsf{P}}$, since the provided compilation runs in polynomial time.

Corollary 3. $\mathsf{N}_{\mathsf{BCI}}^{\mathsf{P},\mathsf{P}}$ *is* PSPACE*-complete.*

Moreover, thanks to Theorem 2 we can go, preserving plan size exactly, from $\mathsf{N}_{\mathsf{BC}}^{\mathsf{P},\mathsf{P}}$ to S_{BC}, and then from S_{BC} to $\mathsf{N}^{\mathsf{P},\mathsf{P}}$, thus compiling away both Boolean conditions and conditional effects from the numerical problem.

Corollary 4. $\mathsf{N}_{\mathsf{BC}}^{\mathsf{P},\mathsf{P}} \preceq^1 \mathsf{N}^{\mathsf{P},\mathsf{P}}$ *and* $\mathsf{N}_{\mathsf{BCI}}^{\mathsf{P},\mathsf{P}} \preceq^1 \mathsf{N}_{\mathsf{I}}^{\mathsf{P},\mathsf{P}}$

Recall from Proposition 1 that removing Boolean conditions and/or conditional effects from classical problems was only possible by growing plan size polynomially, while here we can do so preserving size exactly. This is further evidence, if it was needed at all, of the power of numbers in planning.

We also have an interesting negative consequence of Theorems 2 and 3, namely that incomplete state specifications cannot be compiled away from numeric planning, even preserving plan size polynomially.

Corollary 5. $\mathsf{N}_{\mathsf{I}}^{\mathsf{P},\mathsf{P}} \not\preceq^p \mathsf{N}^{\mathsf{P},\mathsf{P}}$

Proof. Suppose by contradiction that $\mathsf{N}_{\mathsf{I}}^{\mathsf{P},\mathsf{P}} \preceq^p \mathsf{N}^{\mathsf{P},\mathsf{P}}$. Recall that we have $\mathsf{S}_{\mathsf{BI}} \preceq^1 \mathsf{N}_{\mathsf{I}}^{\mathsf{P},\mathsf{P}}$ by Theorem 2, and $\mathsf{N}^{\mathsf{P},\mathsf{P}} \preceq^1 \mathsf{S}_{\mathsf{BC}}$ by Theorem 3. Hence we have:

$$\mathsf{S}_{\mathsf{BI}} \preceq^1 \mathsf{N}_{\mathsf{I}}^{\mathsf{P},\mathsf{P}} \preceq^p \mathsf{N}^{\mathsf{P},\mathsf{P}} \preceq^1 \mathsf{S}_{\mathsf{BC}}$$

This implies $\mathsf{S}_{\mathsf{BI}} \preceq^p \mathsf{S}_{\mathsf{BC}}$, contradicting Proposition 1. □

Note that Corollary 5 tells us that incomplete state specifications are more powerful in numeric planning than in classical planning. In the latter (see Proposition 1), they can be compiled away polynomially, and only when paired with Boolean conditions they make the compilation impossible.

5 Discussion and Conclusions

We considered numeric planning problems with bounds on the value of variables, and studied their compilability to numerical fragments obtained by restricting certain syntactic features, and to classical planning. We based our work on the framework set up by Nebel [2000], where he studied in great details the compilability between classical fragments also in terms of the growth of the length of the compiled plans.

The results, pictured in Figure 1, show a different landscape w.r.t. classical planning. In particular, if we focus on compilability *preserving plan size exactly*, only a few classical fragments are compilable to each other, while in the numeric case, the space is partitioned into two large equivalence classes. The watershed between the two is the *incomplete space specifications*, which cannot be compiled away even with a polynomial increase in plan length. Our results also locate the numeric fragment w.r.t. the classical ones. We saw that numbers can be compiled away (preserving plan size exactly) to a classical problem if we admit Boolean conditions and conditional effects and, *vice versa*, these two features can be compiled away if we admit numbers. In hindsight, this relationship can be explained by a common feature that both numeric and conditional effecs have in common: their outcome is *state-dependent*, meaning that it changes depending on the state where they are applied. This connection was never explicitly quantified before.

We distinguish fragments mainly by the same syntactic features already considered by Nebel [2000], besides the degree of the involved polynomials. However, one could enrich the picture by considering more granularity at the level of the syntax of numeric conditions and effects, *e.g.*, by restricting the kinds of allowed assignments and increments. Our work only focused on bounded numeric planning over discrete variables. A natural extension would be to study numeric fragments over *dense* or *continuous* variables, which invalidate many assumptions used to prove our results. Finally, we only considered polynomial expressions, but PDDL 2.1 supports divisions as well, which may further enrich the picture.

While this work has naturally mostly theoretical interest, it is not devoid of potential practical consequences worth exploring in the future. In particular, the connection between numeric and conditional effects, especially considering that plan length does not change when going from one to the other, can be useful to transfer techniques from one field to the other, e.g., in the definition of new heuristic functions.

Acknowledgements

Nicola Gigante acknowledges the support of the PURPLE project, 1st Open Call for Innovators of the AIPlan4EU H2020 project, a project funded by EU Horizon 2020 research and innovation programme under GA n. 101016442 (since 2021). Enrico Scala is supported directly by the AIPlan4EU H2020 project.

References

[Bertolucci *et al.*, 2019] Riccardo Bertolucci, Alessio Capitanelli, Marco Maratea, Fulvio Mastrogiovanni, and Mauro Vallati. Automated planning encodings for the manipulation of articulated objects in 3d with gravity. In *Proceedings of AI*IA*, pages 135–150, 2019.

[Bozzelli *et al.*, 2020] Laura Bozzelli, Alberto Molinari, Angelo Montanari, Adriano Peron, and Gerhard J. Woeginger. Timeline-based planning over dense temporal domains. *Theor. Comput. Sci.*, 813:305–326, 2020.

[Buss *et al.*, 1992] Samuel R. Buss, Stephen A. Cook, A. Gupta, and V. Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM J. Comput.*, 21(4):755–780, 1992.

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

[Della Monica *et al.*, 2020] Dario Della Monica, Nicola Gigante, Salvatore La Torre, and Angelo Montanari. Complexity of qualitative timeline-based planning. In *Proceedings of the 27th International Symposium on Temporal Representation and Reasoning*, volume 178 of *LIPIcs*, pages 16:1–16:13, 2020.

[Fox and Long, 2003] Maria Fox and Derek Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.*, 20:61–124, 2003.

[Gigante *et al.*, 2020] Nicola Gigante, Andrea Micheli, Angelo Montanari, and Enrico Scala. Decidability and complexity of action-based temporal planning over dense time. In *Proceedings of The Thirty-Fourth AAAI Conference on Artificial Intelligence*, pages 9859–9866. AAAI Press, 2020.

[Haslum and Jonsson, 1999] Patrik Haslum and Peter Jonsson. Some results on the complexity of planning with incomplete information. In Susanne Biundo and Maria Fox, editors, *Proceedings of the 5th European Conference on Planning*, volume 1809 of *Lecture Notes in Computer Science*, pages 308–318. Springer, 1999.

[Haslum *et al.*, 2019] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2019.

[Helmert, 2002] Malte Helmert. Decidability and undecidability results for planning with numerical state variables. In *Proc,. of International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)*, pages 44–53, 2002.

[Hoffmann, 2003] Jörg Hoffmann. The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. *J. Artif. Intell. Res.*, 20:291–341, 2003.

[Jung, 1985] Hermann Jung. Depth efficient transformations of arithmetic into boolean circuits. In Lothar Budach, editor, *Fundamentals of Computation Theory, FCT '85, Cottbus, GDR, September 9-13, 1985*, volume 199 of *Lecture Notes in Computer Science*, pages 167–174. Springer, 1985.

[Kiam *et al.*, 2020] Jane Jean Kiam, Enrico Scala, Miquel Ramírez Javega, and Axel Schulte. An ai-based planning framework for HAPS in a time-varying environment. In *ICAPS*, pages 412–420. AAAI Press, 2020.

[Kuroiwa *et al.*, 2021] Ryo Kuroiwa, Alexander Shleyfman, Chiara Piacentini, Margarita P. Castro, and J. Christopher Beck. Lm-cut and operator counting heuristics for optimal numeric planning with simple conditions. In Susanne Biundo, Minh Do, Robert Goldman, Michael Katz, Qiang Yang, and Hankz Hankui Zhuo, editors, *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling*, pages 210–218. AAAI Press, 2021.

[Nebel, 2000] Bernhard Nebel. On the compilability and expressive power of propositional planning formalisms. *J. Artif. Intell. Res.*, 12:271–315, 2000.

[O'Donnell, 2014] Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.

[Parkinson *et al.*, 2012] Simon Parkinson, Andrew Longstaff, Andrew Crampton, and Peter Gregory. The application of automated planning to machine tool calibration. In *ICAPS*. AAAI, 2012.

[Percassi *et al.*, 2021] Francesco Percassi, Enrico Scala, and Mauro Vallati. Translations from discretised PDDL+ to numeric planning. In *ICAPS*, pages 252–261. AAAI Press, 2021.

[Piacentini *et al.*, 2018] Chiara Piacentini, Margarita P. Castro, André Augusto Ciré, and J. Christopher Beck. Linear and integer programming-based heuristics for cost-optimal numeric planning. In *AAAI*, pages 6254–6261. AAAI Press, 2018.

[Rintanen, 2007] Jussi Rintanen. Complexity of concurrent temporal planning. In Mark S. Boddy, Maria Fox, and Sylvie Thiébaux, editors, *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, pages 280–287. AAAI, 2007.

[Scala *et al.*, 2016] Enrico Scala, Patrik Haslum, Sylvie Thiébaux, and Miquel Ramírez. Interval-based relaxation for general numeric planning. In *Proceedings of ECAI*, volume 285, pages 655–663, 2016.

[Scala *et al.*, 2020] Enrico Scala, Patrik Haslum, Sylvie Thiébaux, and Miquel Ramírez. Subgoaling techniques for satisficing and optimal numeric planning. *J. Artif. Intell. Res.*, 68:691–752, 2020.

[Vallati *et al.*, 2016] Mauro Vallati, Daniele Magazzeni, Bart De Schutter, Lukás Chrpa, and Thomas Leo McCluskey. Efficient macroscopic urban traffic models for reducing congestion: A PDDL+ planning approach. In *Proceedings of AAAI*, pages 3188–3194, 2016.

[Vollmer, 1999] Heribert Vollmer. *Introduction to Circuit Complexity - A Uniform Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999.

[von zur Gathen and Seroussi, 1991] Joachim von zur Gathen and Gadiel Seroussi. Boolean circuits versus arithmetic circuits. *Inf. Comput.*, 91(1):142–154, 1991.

[Wegener, 1987] Ingo Wegener. *The complexity of Boolean functions*. Wiley-Teubner, 1987.