Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)

Planning over Integers: Compilations and Undecidability

Daniel Gnad[1], **Malte Helmert**[2], **Peter Jonsson**[1], **Alexander Shleyfman**[3]

[1]Department of Computer and Information Science, Linköping University, Linköping, Sweden
[2] Department of Mathematics and Computer Science, University of Basel, Basel, Switzerland
[3]Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
{daniel.gnad, peter.jonsson}@liu.se, malte.helmert@unibas.ch, alexander.shleyfman@biu.ac.il

Abstract

Restricted Tasks (RT) are a special case of numeric planning characterized by numeric conditions that involve one numeric variable per formula and numeric effects that allow only the addition of constants. Despite this, RTs form an expressive class whose planning problem is undecidable. The restricted nature of RTs often makes problem modeling awkward and unnecessarily complicated. We show that this can be alleviated by compiling mathematical operations that are not natively supported into RTs using macro-like action sequences. With that, we can encode many features found in general numeric planning such as constant multiplication, addition of linear formulas, and integer division and residue. We demonstrate how our compilations can be used to capture challenging mathematical problems such as the (in)famous *Collatz conjecture*. Our approach additionally gives a simple undecidability proof for RTs, and the proof shows that the number of variables needed to construct an undecidable class of RTs is surprisingly low: two numeric and one propositional variable.

Introduction

Planning with numeric fluents is undecidable even under severe restrictions (Helmert 2002). Nevertheless, significant progress has been made in developing planners suitable for numeric tasks (Scala, Haslum, and Thiébaux 2016; Kuroiwa et al. 2021). The scope of applicability of such planners is not well-understood but it is safe to assume that they are more efficient when used for highly restricted planning formalisms. Hence, it is interesting to study formalisms such as Restricted Tasks (RT) by Hoffmann (2003), where numeric conditions only involve one numeric variable per formula and numeric effects only allow the addition of constants. Even though RT is an extremely restricted formalism, its planning problem is still undecidable.

Modelling problems in RT is complicated, though, which leads to several drawbacks. The most obvious one is the difficulty of handling natural problems within the formalism. Even though one's ambition may not be to fully capture complex problems such as industrial systems, it is still beneficial to understand the expressivity of RT. This may, for instance, be used when exploiting RT as a limited formalism for heuristic computation, e. g. to combine improved computational properties with maximal possible expressivity. Another example is the construction of benchmark problems where one wants to combine simplicity (and consequently understandability) with adverse computational properties.

We investigate the expressivity of RT via task compilations that use *macro-like* action sequences to encode unsupported mathematical operations. Compilations are a means to express features of a richer formalism, such as axioms, conditional effects, or state-dependent action cost, into a simpler one (Nebel 2000; Thiébaux, Hoffmann, and Nebel 2005; Speck et al. 2021). Macros are a common tool for representing subsequences of operators, and have proven to be useful in various respects (Korf 1987; Giménez and Jonsson 2008; Chrpa 2010; Chrpa and Vallati 2022). They are usually considered to be compound actions that replace the original component actions. We use the term here informally to refer to sequences of actions that have to be applied in a given order. Our mathematical macros compile complex numeric operations into the restricted language available in RT.

We assume all numbers to be integers and first introduce a formalism that can represent linear expressions in a straightforward way. By partially using ideas from Hoffmann (2003), we show that tasks in this formalism can be compiled into RT in a way that suggests macro-like constructions for the underlying mathematical operations. This is done by introducing additional variables for controlling the action execution. One consequence of these results is that RT and a particular numeric planning formalism INT have exactly the same expressibility up to polynomial-time reducibility. We emphasize that INT is able to handle a broad range of arithmetic operations—INT is a natural numeric planning subclass of PDDL2.1 level 2 (Fox and Long 2003).

Mathematical macros add a whole new repertoire of powerful and easy-to-use operations to RT. We demonstrate this by modelling the computational problem underlying the *Collatz conjecture* in RT. The Collatz conjecture is one of the most well-known unsolved problems in mathematics (Lagarias 2010); Paul Erdős famously stated that "Mathematics may not be ready for such problems". The representation of this problem becomes straightforward and highly understandable in RT via the compilations, illustrating that mathematical modelling in RT is significantly simplified. Our approach also allows us to provide a simple undecidability

proof for RT and it shows that the required number of variables is notably small—only two numeric and one propositional variable. The result holds even if every action affects at most one numeric variable, i.e. in the absence of numeric cross-effects. This adds new pieces to our understanding of the complexity landscape of numeric planning.

Restricted Tasks

We start with a fragment of numeric planning based on the finite-domain planning (FDR) formalism (Bäckström and Nebel 1995; Helmert 2009) augmented with numeric fluents. We focus on a variant of numeric planning defined by Hoffmann (2003) called the *restricted numeric task* (RT).

Formally, we define an RT as a tuple $\Pi^{\text{RT}} = \langle \mathcal{V}, \mathcal{A}, s_0, G \rangle$. Here, $\mathcal{V} = \mathcal{V}_n \cup \mathcal{V}_p$ is a set of *numeric* and *propositional variables*, respectively. The set $\mathcal{A}$ is the actions of the task, s_0 is the *initial state*, and G is the set of *goal conditions*. We assume all these sets to be finite. Each variable $v \in \mathcal{V}_p$ has a finite *domain* $\mathcal{D}(v)$ and each variable $v \in \mathcal{V}_n$ can take any rational value, i.e., $\mathcal{D}(v) = \mathbb{Q}$. The *states* of Π^{RT} form a set $\mathcal{S} := \times_{v \in \mathcal{V}_p} \mathcal{D}(v) \times \times_{v \in \mathcal{V}_n} \mathbb{Q}$, corresponding to the possible assignments to $\mathcal{V}$. A *state* $s \in \mathcal{S}$ is a tuple $\langle s_p, s_n \rangle$, where $s_p \in \times_{v \in \mathcal{V}_p} \mathcal{D}(v)$ and $s_n \in \times_{v \in \mathcal{V}_n} \mathbb{Q}$. $\langle v, d \rangle$ denotes a *fact*, where $v \in \mathcal{V}$ and $d \in \mathcal{D}(v)$. We say that $s \models (v = d)$ iff $\langle v, d \rangle \in s$, and write $s[v] = d$, i.e., $s[v]$ indicates the value of $v \in \mathcal{V}$ in state s. A state s can be seen as a set of facts so we write $s = s_p \cup s_n$ as a minor abuse of notation. A set of facts s^{pt} is a *partial state* if $s' \subseteq s$ and $s \in \mathcal{S}$.

Conditions can be either propositional, i. e., partial propositional states s^{pt}, or numeric, defined as $v \bowtie w$, with $\bowtie \in \{>, \geq, <, \leq\}$, $v \in \mathcal{V}_n$, and $w \in \mathbb{Q}$. A propositional condition ψ is satisfied by the state s if $s^{pt} \subseteq s$. A numeric condition $v \bowtie w$ is satisfied by s if $s[v] \bowtie w$.

An action $a \in \mathcal{A}$ is a tuple $\langle \mathsf{pre}(a), \mathsf{eff}(a) \rangle$, where $\mathsf{pre}(a)$ is the *precondition* and $\mathsf{eff}(a)$ the *effect* of a. Preconditions are given by $\mathsf{pre}_p(a) \cup \mathsf{pre}_n(a)$, propositional and numeric conditions, respectively. *Effects* $\mathsf{eff}(a) := \mathsf{eff}_p(a) \cup \mathsf{eff}_n(a)$ are similarly defined as sets of propositional and numeric effects. We assume throughout the paper that each action has at most one effect on each variable. In RT, numeric effects have the form $(v \mathrel{+}= c)$, where $v \in \mathcal{V}_n$ and $c \in \mathbb{Q} \setminus \{0\}$. An action a is *applicable* in state s if $s \models \mathsf{pre}(a)$. The result of applying a in s is denoted by $s[\![a]\!] := s'_p \cup s'_n$, with $s'_p[v] = d$ if $\langle v, d \rangle \in \mathsf{eff}_p(a)$, $s'_n[v] = s_n[v] + c$ if $(v \mathrel{+}= c) \in \mathsf{eff}_n(a)$, and $s[\![a]\!][v] = s[v]$ otherwise.

The goal condition $G = G_p \cup G_n$ is a set of propositional and numeric conditions, respectively. s_* is a *goal state* if $s_* \models G$. A *plan* is a consecutively applicable action sequence π that start in s_0 and ends in some $s_* \models G$.

Each RT can be transformed into its integer form, where all constants are in $\mathbb{Z}$ (Helmert 2003).

Planning over Integers

In this section we present a formalism that extends the arithmetic operations supported by restricted tasks. Our formalism assumes all numbers to be integers and allows much broader options for planning over integer numbers. We base our definition on a subset of numeric planning tasks as defined in PDDL2.1 level 2 (Fox and Long 2003), introducing additional features to the formalism.

Let an Integer Numeric Planning Task (INT) be a tuple $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, G \rangle$. The propositional part of the task is represented in FDR as before. The numeric part of the task is defined as follows. We assume that all numeric variables $\mathcal{V}_n$ have integer values, so $s_0[x] \in \mathbb{Z}$ for all $x \in \mathcal{V}_n$. For numeric conditions, we allow formulas of the form $\sum_{x \in \mathcal{V}_n} w_x x \bowtie w_0$, where $\bowtie \in \{>, \geq, <, \leq\}$, and $w_x, w_0 \in \mathbb{Z}$. We say that $s \models \sum_{x \in \mathcal{V}_n} w_x x \bowtie w_0$ if it holds that $\sum_{x \in \mathcal{V}_n} w_x s[x] \bowtie w_0$.[1]

The numeric effects are more intricate. Unlike RTs that allow only numeric effects of the form $x \mathrel{+}= c$, we introduce effects of the form $x := \xi$, where ξ is the linear formula $\sum_{x \in \mathcal{V}_n} c^{\xi}_x x + c^{\xi}_0$, with integer coefficients $c^{\xi}_x, c^{\xi}_0 \in \mathbb{Z}$. This subsumes effects $x \mathrel{+}= c$, encoding them as $x := x + c$.

We introduce two additional operators: least positive residue $\bmod$, and integer division $//$. If $a, c \in \mathbb{Z}$, then there are two numbers $q \in \mathbb{Z}$ and $r \in \mathbb{N}_0$, s.t. $a = c \cdot q + r$, and $r < |q|$; we write $q = a//c$ and $r = a \bmod c$. With this, we allow two novel numeric action effects in INT: $x := y//c$ and $x := y \bmod c$, where $x, y \in \mathcal{V}_n$ and $c \in \mathbb{Z}$.

From INT to RT in Polynomial Time

It is obvious that any RT can be efficiently transformed into an equivalent INT: every RT can be converted into an integer RT (as pointed out earlier) and each effect $x \mathrel{+}= c$ can be replaced with $x := x + c$. In the following, we will prove that any INT can be efficiently transformed into an RT. We start by transforming all linear conditions into restricted conditions over one variable, as described by Hoffmann (2003).

Every linear condition $\psi = \sum_{x \in V} w^{\psi}_x x \bowtie w_0$ that appears in INT, can be brought to some canonical integer linear normal form (ILNF) by dividing it by the GCD of all constants involved, and assuring that the first variable in a predefined order is always positive (cf. LNF in Hoffmann 2003). For every condition ψ we then introduce a variable y^{ψ}, and replace all occurrences of $\psi \bowtie w_0$ with $y^{\psi} \bowtie w_0$.

We need to assure that in every state s it holds that $s[y^{\psi}] = \sum_{x \in V} w^{\psi}_x s[x]$. This is done by initializing all new variables to $s_0[y^{\psi}] = \sum_{x \in V} w^{\psi}_x s_0[x]$, and adding an effect:

$$y^{\psi} := \sum_{x \in V} w^{\psi}_x \xi_x,$$

to each action a that affects at least one of the variables in $V \subseteq \mathcal{V}_n$. Here, either $(x := \xi_x) \in \mathsf{eff}_n(a)$ or $\xi_x = x$ if the variable x is not affected by a. The effect on y^{ψ} is well-defined because every action has at most one effect on each numeric variable x.

It remains to show that a weighted sum of the effects of the form $y := cx$, residue $y := x \bmod c$, and division $y := x//c$ can be encoded using only propositional variables and effects of the form $y \mathrel{+}= c$, where $x, y \in \mathcal{V}_n$ and $c \in \mathbb{Z}$. We do so in the next section by introducing macro-like constructions that compile these operations into RT.

[1] Note that any linear formula can be brought to an integer form by multiplication with the GCD of all denominators.

Compiling Integer Arithmetics into RT

We compile away the aforementioned new arithmetic operations by introducing mathematical macros, i. e., sequences of actions, that only employ the basic operations supported by RTs. We do so by adding new variables and actions to organize the control flow and store interim results. We remark that we are merely scratching the surface of possible operations, and statements such as exponentiation, absolute value, and many others can be compiled into RT in a similar way.

For every compilation scheme, we introduce a propositional variable ctrl that controls the execution, each of which has the value $\bot$, and a value per component action. To avoid that intermediate states satisfy the goal condition, we add the fact $\langle \mathrm{ctrl}, \bot \rangle$ to the initial state s_0 and goal G. The variable ctrl is set to the value of the current macro action executed, and returns to be $\bot$ at the end of the execution of a macro. The compilations, as defined next, are correct only if there is a single respective integer effect per action. We discuss how to support multiple effects at the end of the section.

Assignment: Let a be an action with numeric effects that include $\xi : (x := c)$ for $x \in \mathcal{V}_n, c \in \mathbb{Z}$:

We introduce a propositional variable v_{xC} with domain $\mathcal{D}(v_{xC}) = \{\bot\} \cup \{c \mid \exists a : (x := c) \in \mathsf{eff}_n(a)\}$. We change the effect of a by removing ξ and adding a propositional effect $\langle v_{xC}, c \rangle$. Furthermore, we add actions that incrementally set the value of x to c as follows (we show the actions for the case where $x < c$, the case where $x > c$ works analogously):

- $a_{x:=C}$ with $\mathsf{pre}(a_{x:=C}) = \{\langle v_{xC}, c \rangle\} \cup \{x < c\}$ and $\mathsf{eff}(a_{x:=C}) = \{x \mathrel{+}= 1\}$,
- $a_{x=C}$ with $\mathsf{pre}(a_{x=C}) = \{\langle v_{xC}, c \rangle\} \cup \{x = c\}$ and $\mathsf{eff}(a_{x=C}) = \{v_{xC} = \bot\}$.

Finally, we add a precondition $\langle v_{xC}, \bot \rangle$ to all existing actions ensuring that the macro is completed after applying a and no other action is applicable in between. Note that all assignment effects to x are encoded in a single propositional variable v_{xC}. This can be generalized by using v_{xC} for assignments to any numeric variable $y \in \mathcal{V}_n$ by "remembering" the variable to be assigned to in the value of v_{xC}.

Multiplication by a constant: Let a be an action with numeric effects including $\xi : (y := cx)$ for $x, y \in \mathcal{V}_n, c \in \mathbb{Z}$:

We introduce a propositional variable v_{ycx} with domain $\mathcal{D}(v_{ycx}) = \{\bot, \mathrm{buf}, \mathrm{mul}\}$ and a numeric buffer variable b. We change the effect of a by removing ξ and adding effects $\{\langle v_{ycx}, \mathrm{buf} \rangle\} \cup \{(y := 0), (b := 0)\}$, and introduce new actions that first buffer the value of x and then incrementally set y to the value cx as follows:

- $a_{\mathrm{buf}x}$ with $\mathsf{pre}(a_{\mathrm{buf}x}) = \{\langle v_{ycx}, \mathrm{buf} \rangle\} \cup \{x > 0\}$ and $\mathsf{eff}(a_{\mathrm{buf}x}) = \{(x \mathrel{-}= 1), (b \mathrel{+}= 1)\}$,
- $a_{b=x}$ with $\mathsf{pre}(a_{b=x}) = \{\langle v_{ycx}, \mathrm{buf} \rangle\} \cup \{x = 0\}$ and $\mathsf{eff}(a_{b=x}) = \{\langle v_{ycx}, \mathrm{mul} \rangle\}$,
- $a_{y:=cx}$ with $\mathsf{pre}(a_{y:=cx}) = \{\langle v_{ycx}, \mathrm{mul} \rangle\} \cup \{\mathrm{buf} > 0\}$ and $\mathsf{eff}(a_{y:=cx}) = \{(x \mathrel{+}= 1), (b \mathrel{-}= 1), (y \mathrel{+}= c)\}$,
- $a_{y=cx}$ with $\mathsf{pre}(a_{y=cx}) = \{\langle v_{ycx}, \mathrm{mul} \rangle\} \cup \{\mathrm{buf} = 0\}$ and $\mathsf{eff}(a_{y=cx}) = \{\langle v_{ycx}, \bot \rangle\}$.

As before, for $x < 0$ or $c < 0$ we need analogous actions, and to ensure that the macro is completed after applying a, we add the precondition $\langle v_{ycx}, \bot \rangle$ to all existing actions. A single propositional variable can be used to compactly encode such kinds of effects for all numeric variables.

We remark that this compilation can be generalized to effects that are linear weighted sums over several numeric variables by sequentially executing it for all parts of the sum.

Least positive residue: Let a be an action with numeric effects that include $\xi : (x := x \bmod c)$ for $x \in \mathcal{V}_n, c \in \mathbb{Z}$: We introduce a propositional variable $v_{x \bmod C}$ with domain $\mathcal{D}(v_{x \bmod C}) = \{\bot\} \cup \{c \mid \exists a : (x := x \bmod c) \in \mathsf{eff}_n(a)\}$. We change the effect of a by removing ξ and adding a propositional effect $\langle v_{x \bmod C}, c \rangle$ and add new actions that deduct c from x until it has the value $x \bmod c$ as follows (for $x > 0, c > 0$):

- $a_{x \bmod C}$ with $\mathsf{pre}(a_{x \bmod C}) = \{\langle v_{x \bmod C}, c \rangle\} \cup \{x \geq c\}$ and $\mathsf{eff}(a_{x \bmod C}) = \{x \mathrel{-}= c\}$,
- $a_{x \bmod \bot}$ with $\mathsf{pre}(a_{x \bmod \bot}) = \{\langle v_{x \bmod C}, c \rangle\} \cup \{x < c\}$ and $\mathsf{eff}(a_{x \bmod \bot}) = \{v_{x \bmod C} = \bot\}$.

The cases where x and/or c are negative can be handled in the same way so there are four cases in total. We add the precondition $\langle v_{x \bmod C} = \bot \rangle$ to all existing actions as before.

Note that the effect $\xi : (y := x \bmod c)$ can be modeled using consecutively $y := x$ and $y := y \bmod c$.

Integer division: Let a be an action with numeric effects that include $\xi : (x := x//c)$ for $x \in \mathcal{V}_n, c \in \mathbb{Z}$:

We introduce a propositional variable $v_{x//C}$ with domain $\mathcal{D}(v_{x//C}) = \{\bot\} \cup \{c \mid \exists a : (x := x//c) \in \mathsf{eff}_n(a)\}$ and a numeric buffer variable b. We change the effect of a by removing ξ and adding effects $\{\langle v_{x//C}, c \rangle\} \cup \{(b := 0)\}$, and add new actions that incrementally compute $x//c$ in the buffer and then set x to the value of b as follows:

- $a_{x//C}$ with $\mathsf{pre}(a_{x//C}) = \{\langle v_{x//C}, c \rangle\} \cup \{x \geq c\}$ and $\mathsf{eff}(a_{x//C}) = \{(x \mathrel{-}= c), (b \mathrel{+}= 1)\}$,
- $a_{x//\bot}$ with $\mathsf{pre}(a_{x//\bot}) = \{\langle v_{x//C}, c \rangle\} \cup \{x < c\}$ and $\mathsf{eff}(a_{x//\bot}) = \{v_{x//C} = \bot\} \cup \{x := b\}$.

As before, we add guarding preconditions $\langle v_{x//c}, \bot \rangle$ to all existing actions, and remark that the cases where x and/or c are negative can be modelled in the same way.

Linear weighted sums: Let a be an action with numeric effects that include $\xi : (x := \sum_{y \in V} w_y \xi_y)$ for $V \subseteq \mathcal{V}_n, x \in \mathcal{V}_n, w_y \in \mathbb{Z}$. We add a propositional variable v_Σ, where $\mathcal{D}(v_\Sigma) = \{\bot\} \cup \{\mathrm{buf}_y, \mathrm{add}_y \mid y \in V\}$, and for each variable $y \in V$ a numeric variable b_y. The new variables are initialized to $\langle v_\Sigma, \bot \rangle$ and $b_y = 0$ for each $y \in V$. We change the effect of a by removing ξ and adding the effect $\{\langle v_\Sigma, \mathrm{buf}_y \rangle\}$. Assume some order on the variables in V:

- Let ξ_y be one of the following effects cy, $y \bmod c$ or $y//c$. Using the above compilation schemes, we assign $b_y := \xi_y$, and set v_Σ to the next variable in the order of V until all buffers at initialized.
- When all buffers are set, we set $x := 0$ and change the value of v_Σ to the first $\{\langle v_\Sigma, \mathrm{add}_y \rangle\}$ according to the given order of variables.

- Last, in the given variable order, we add $x \mathrel{+}= w_y b_y$ and set $b_y := 0$ for all $y \in V$, and finally set v_Σ to $\bot$.

This concludes our introduction of compilations for integer arithmetics. Note that for the macros to be applicable in the same plan, each action has to have preconditions of the form $\{\langle \mathrm{ctrl}, \bot\rangle\}$ for each propositional flag variable ctrl of all other action compilations involved.

Application of multiple numeric effects: Note that sometimes the application of multiple numeric effects of an action may lead to inconsistency. Consider, for example, an action a with two numeric effects $x := y$ and $y := x$, where application of one of the effects may hurt the application of the other. To work around this problem, we introduce one additional buffer per affected variable. In this example, these are temp_x and temp_y. Then, we can use the standard trick to swap between variables. We use macros to ensure the following sequence of assignments: we first set the buffers $\mathrm{temp}_x := x$, $\mathrm{temp}_y := y$, and then use the buffers to assign the values to the original variables $x := \mathrm{temp}_y$ and $y := \mathrm{temp}_x$. We remark that multiple action effects can be handled in the same way for all compilations we introduced.

Collatz Problems

We will next exemplify how our compilations can be employed in a more intricate setting. Consider the following function on positive integers:

$$f(n) = \begin{cases} n/2, & n \text{ is even} \\ 3n+1, & n \text{ is odd} \end{cases}$$

Lothar Collatz conjectured that for every $n \in \mathbb{N}$ there is a number k such that $f^{(k)}(n) = 1$, where $f^{(k)}$ denotes k-times recursive application of f. This conjecture is one of the most famous open problems in mathematics (Lagarias 2010). Conway (1972) introduced *Collatz functions* as follows. Choose an integer p together with rational numbers $a_i, b_i \in \mathbb{Q}$, $0 \leq i \leq p-1$. A function g is *Collatz* if $g(n) = a_i n + b_i$ is integral whenever $n = i \bmod p$. We see that the function f above is Collatz by choosing $p = 2$, $a_0 = 1/2$, $b_0 = 0$, $a_1 = 3$, and $b_1 = 1$. We show that for every Collatz function g, there is an RT that gets n as an input in its initial state and reaches its goal state if and only if there is a $k \in \mathbb{N}$ s.t. $g^{(k)}(n) = 1$. This is not purely a modelling exercise: Conway (1987) proved the following undecidability result.

Theorem 1 (Conway 1987)**.** *There is a Collatz function g such that the following problem is undecidable: given a positive integer n, does there exist a positive integer k such that $g^{(k)}(n) = 1$?*

Thus, our transformation into RT immediately implies undecidability of RT planning. We will now present the details of the transformation. Let g be a Collatz function and let $a_i, b_i \in \mathbb{Q}$ and $0 \leq i \leq p-1$ be the numbers that define g. The variables of our RT are $\mathcal{V}_n = \{x, \mathrm{buff}_x\}$ and $\mathcal{V}_p = \{\mathrm{ctrl}\}$, where $\mathcal{D}(\mathrm{ctrl}) = \{\bot, 1\mathrm{mod}_p, \ldots, (p-1)\mathrm{mod}_p\}$. The initial state s_0 is $\{\langle x, n\rangle, \langle \mathrm{buff}_x, 0\rangle, \langle \mathrm{ctrl}, \bot\rangle\}$ and the goal condition G is $\{x = 1, \mathrm{buff}_x = 0\} \cup \{\langle \mathrm{ctrl}, \bot\rangle\}$. The actions first compute $x \bmod p$ and then do a case distinction based on the residue values as follows:

- a_{mod_p} with $\mathsf{pre}(a_{\mathrm{mod}_p}) = \{x \geq p\} \cup \{\langle \mathrm{ctrl}, \bot\rangle\}$ and $\mathsf{eff}(a_{\mathrm{mod}_p}) = \{x \mathrel{-}= p, \mathrm{buff}_x \mathrel{+}= 1\}$. Note that this also sets $\mathrm{buff}_x := x//p$ along application.
- For each $i \in [p] - 1$ we define three types of actions:
  - a_{set_i} makes the case distinction:
    $\mathsf{pre}(a_{\mathrm{set}_i}) = \{x = i\} \cup \{\langle \mathrm{ctrl}, \bot\rangle\}$, and
    $\mathsf{eff}(a_{\mathrm{set}_i}) = \{\langle \mathrm{ctrl}, i\mathrm{mod}_p\rangle\} \cup \{x \mathrel{+}= (a_i - 1)i\}$.
  - a_{mult_i} multiplies x by a_i:
    $\mathsf{pre}(\mathrm{mult}_i) = \{\mathrm{buff}_x > 0\} \cup \{\langle \mathrm{ctrl}, i\mathrm{mod}_p\rangle\}$, and
    $\mathsf{eff}(\mathrm{mult}_i) = \{x \mathrel{+}= a_i p, \mathrm{buff}_x \mathrel{-}= 1\}$.
  - a_{add_i} adds b_i to x and finishes the macro:
    $\mathsf{pre}(\mathrm{add}_i) = \{\mathrm{buff}_x = 0\} \cup \{\langle \mathrm{ctrl}, i\mathrm{mod}_p\rangle\}$, and
    $\mathsf{eff}(\mathrm{add}_i) = \{x \mathrel{+}= b_i\} \cup \{\langle \mathrm{ctrl}, \bot\rangle\}$.

It is easy to see that this exactly encodes the Collatz problem for any function g and input value n; the value of $g^k(n)$ is achieved iff $\mathrm{buff}_x = 0$ and $\langle \mathrm{ctrl}, \bot\rangle$. Thus, the ability to encode integer division and residue in RT, allows us to compactly model well-known mathematical problems.

Finally, we remark that, while the above encoding of Collatz problems has actions with effects on both numeric variables, we can easily avoid such cross-effects, so that every action affects at most one numeric variable. This is achieved by introducing additional propositional values to sequentialize cross-effects. For a_{mult_i}, we split every value $i\mathrm{mod}_p$ of ctrl into two, replace the effect $(\mathrm{buff}_x \mathrel{-}= 1)$ by $\langle \mathrm{ctrl}, i\mathrm{mod}_p^1\rangle$, and add an action with precondition $\{\langle \mathrm{ctrl}, i\mathrm{mod}_p^1\rangle\}$ and effects $\{\langle \mathrm{ctrl}, i\mathrm{mod}_p^1\rangle, (\mathrm{buff}_x \mathrel{-}= 1)\}$.

This means that, coming as a surprise to us, RTs are undecidable for only two numeric and one propositional variable, even in the absence of numeric cross-effects.

Conclusions and Research Directions

We have demonstrated how to compile various not natively supported mathematical operations into RT. This allows us to model numeric planning problems in a much more straightforward way than by working in the basic RT formalism. A natural question in this context is what the limits of such compilations are: can we formally characterize the mathematical operations that are compilable to RT?

Our results show that RT and INT have the same expressive power in a precise sense: the formalisms are equivalent up to polynomial-time preprocessing of planning instances. Moreover, we show that RT is equivalent to linear numeric tasks (Hoffmann 2003), where all constants are integers. Analysing and comparing the features of various planning formalisms has a long history in planning: one of the first examples of a systematic approach was presented by Nebel (2000) and there is now a row of related results in the literature, cf. the recent papers by Lin and Bercher (2022) and Scheck, Niveau, and Zanuttini (2021). Most of the results presented in the literature are concerned with finite-domain planning so our result is different in this respect. Thus, analysing and comparing the expressive power of numeric planning appears to be a feasible research direction.

Acknowledgments

Daniel Gnad was partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215, and by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Peter Jonsson was partially supported by the Swedish Research Council (VR) under grant 2021-0437.

References

Bäckström, C.; and Nebel, B. 1995. Complexity results for SAS^+ planning. *Computational Intelligence*, 11(4): 625–655.

Chrpa, L. 2010. Generation of macro-operators via investigation of action dependencies in plans. *Knowledge Engineering Review*, 25(3): 281–297.

Chrpa, L.; and Vallati, M. 2022. Planning with critical section macros: theory and practice. *Journal of Artificial Intelligence Research*, 74: 691–732.

Conway, J. H. 1972. Unpredictable iterations. In *Number Theory Conference*, 49–52. University of Colorado.

Conway, J. H. 1987. Fractran, a simple universal computing language for arithmetics. In *Open Problems in Communication and Computation*, 3–27. Springer Verlag.

Fox, M.; and Long, D. 2003. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20: 61–124.

Giménez, O.; and Jonsson, A. 2008. The complexity of planning problems with simple causal graphs. *Journal of Artificial Intelligence Research*, 31: 319–351.

Helmert, M. 2002. Decidability and undecidability results for planning with numerical state variables. In *Proc. 6th International Conference on Automated Planning and Scheduling (AIPS-2002)*, 303–312.

Helmert, M. 2003. Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, 143(2): 219–262.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173: 503–535.

Hoffmann, J. 2003. The Metric-FF planning system: translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*, 20: 291–341.

Korf, R. E. 1987. Planning as search: a quantitative approach. *Artificial Intelligence*, 33(1): 65–88.

Kuroiwa, R.; Shleyfman, A.; Piacentini, C.; Castro, M. P.; and Beck, J. C. 2021. LM-cut and operator counting heuristics for optimal numeric planning with simple conditions. In *Proc. 31st International Conference on Automated Planning and Scheduling (ICAPS-2021)*, 210–218.

Lagarias, J. C. 2010. *The Ultimate Challenge: The* $3x + 1$ *Problem*. American Mathematical Society.

Lin, S.; and Bercher, P. 2022. On the expressive power of planning formalisms in conjunction with LTL. In *Proc. 32nd International Conference on Automated Planning and Scheduling (ICAPS-2022)*, 231–240.

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research*, 12: 271–315.

Scala, E.; Haslum, P.; and Thiébaux, S. 2016. Heuristics for numeric planning via subgoaling. In *Proc. 25th International Joint Conference on Artificial Intelligence (IJCAI-2016)*, 3228–3234.

Scheck, S.; Niveau, A.; and Zanuttini, B. 2021. Knowledge compilation for nondeterministic action languages. In *Proc. 21st International Conference on Automated Planning and Scheduling (ICAPS-2021)*, 308–316.

Speck, D.; Borukhson, D.; Mattm"uller, R.; and Nebel, B. 2021. On the Compilability and Expressive Power of State-Dependent Action Costs. In Goldman, R. P.; Biundo, S.; and Katz, M., eds., *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*, 358–366. AAAI Press.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In Defense of PDDL Axioms. *AIJ*, 168(1–2): 38–69.