Complexity Issues of Interval Relaxed Numeric Planning

Johannes Aldinger and Robert Mattmüller and Moritz Göbelbecker
Albert-Ludwigs-Universität, Institut für Informatik
79110 Freiburg, Germany
{aldinger,mattmuel,goebelbe}@informatik.uni-freiburg.de

Abstract
Automated planning is a hard problem even in its most basic form as STRIPS planning. We are interested in numeric planning tasks with instantaneous actions, a problem which is not even decidable in general. Relaxation is an approach to simplifying complex problems in order to obtain guidance in the original problem. We present a relaxation approach with intervals for numeric planning and discuss the arising complexity issues.

Introduction
Relaxation is a predominant approach to simplifying planning problems. Solutions of the relaxed planning problem can be used to guide search in the original planning task. The forward propagation heuristic \( h_{\text{add}} \) (Bonet, Loerincs, and Geffner 1997; Bonet and Geffner 2001) was used in the heuristic search planner that won the first International Planning Competition (IPC 1998) and \( h_{\text{max}} \) (Bonet and Geffner 1999) is its admissible counterpart. The underlying assumption of a delete relaxation is that propositions which are achieved once during planning can not be invalidated. More recent planning systems are usually not restricted to propositional state variables of the planning problem. Instead they use the \( \text{SAS}^+ \) formalism (Bäckström and Nebel 1993) which allows for (finite-domain) multi-valued variables. Unlike propositional STRIPS (Fikes and Nilsson 1971), a “delete relaxation” corresponds to variables that can attain a set of values at the same time. Extending this concept for numeric planning relaxes the set representation even further. Numeric variables can have infinitely many values which makes it impossible to store all of them. A memory efficient approach is to consider the enclosing interval of all possible values for each numeric variable. The methods to deal with intervals have been subject to the field of interval arithmetic for decades (Moore, Kearfott, and Cloud 2009) and enables us to deal with intervals in terms of basic numeric operations.

Numeric planning tasks can require actions to be applied multiple times, as setting a numeric variable to a target value can require multiple steps even in relaxed problems. In this paper we provide the foundations for interval relaxed numeric planning.

Related Work
Extending the concept of classical planning heuristics to numeric problems has been done before, albeit only for a subset of numeric tasks. In many relevant real world problems, numeric variables can only be manipulated in a restricted way. The Metric-FF planning system (Hoffmann 2003) tries to convert the planning task into a linear numeric task which ensures that variables can “grow” in only one direction. By introducing inverted auxiliary variables for decreasing numeric variables, the concept of a delete relaxation translates into a relaxation where decrease effects are considered harmful and higher values of a variable are always beneficial to fulfill the preconditions of actions.

More recently, Coles et al. (2008) investigated an approach based on linear programs. In many relevant real world applications, numeric variables are used to model resources. Delete relaxation heuristics fail to offer guidance on such problems if a cyclic resource transfer is possible. As delete relaxations make the assumption that sub-goals stay achieved, a resource transfer can “produce” resources without decreasing them at their original destination. Coles et al. analyze the planning problem for consumers and producers of resources and build a linear program to ensure that resources are not more often consumed than produced or initially available to obtain an informative heuristic.

Basics
In this section we outline numeric planning with instantaneous actions which is expressible in PDDL2.1, layer 2 (Fox and Long 2003). We present an overview over interval arithmetic, the technique we use to extend delete relaxation heuristics to numeric planning. The section closes with a short complexity discussion.

Numeric Planning with Instantaneous Actions
Given a set of variables \( \mathcal{V} \) with domains \( \text{dom}(v) \) for all \( v \in \mathcal{V} \), a state \( s \) is a mapping of variables \( v \) to their respective domains. Throughout the paper, we denote the value of a variable \( v \) in a state \( s \) by \( s(v) \).

A numeric planning task \( \Pi = (\mathcal{V}_p, \mathcal{V}_N, \mathcal{O}, \mathcal{I}, \mathcal{G}) \) is a 5-tuple where \( \mathcal{V}_p \) is a set of propositional variables \( v_p \) with domain \{true,false\}, \( \mathcal{V}_N \) is a set of numeric variables \( v_n \) with domain \( \mathbb{Q}^\infty \) where we abbreviate \( \mathbb{Q}^\infty \) for
A propositional effect \( o \) is a numeric effect \( s \), and \( o \in \{ +, -, \times, \div \} \) and expressions \( e_1 \) and \( e_2 \) recursively defined over variables \( V \) and constants from \( \mathbb{Q} \). A numeric constraint \( \text{con} = (e_1 \circ e_2) \) compares numeric expressions \( e_1 \) and \( e_2 \) with \( \in \{ \leq, <, =, \neq, \} \). A condition is a conjunction of propositions and numeric constraints. A numeric effect is a triple \( (v_n, o, e) \) where \( v_n \in V_N \), \( o \in \{ +, -, \times, \div \} \), and \( e \) is a numeric expression. Operators \( o \in \mathcal{O} \) are of the form \( \{ \text{pre} \to \text{eff} \} \) and consist of a pre-condition \( \text{pre} \) and a set of effects \( \text{eff} \) which contains at most one numeric effect for each numeric variable \( v_n \) and at most one truth assignment for each propositional variable \( v_p \).

The semantic of a numeric planning task is straightforward. For constants \( c \in \mathbb{Q} \), \( s(c) = c \) by abuse of notation. Numeric expressions \( (e_1 \circ e_2) \) for \( o \in \{ +, -, \times, \div \} \) are recursively evaluated in state \( s \): \( s(e_1 \circ e_2) = s(e_1) \circ s(e_2) \). A state satisfies a condition \( s \models \rho_p \) iff \( s(v_p) = \text{true} \), where \( v_p \in V_P \). For numeric constraints, \( s \models (e_1 \circ e_2) \) iff \( s(e_1) \circ s(e_2) \), where \( \in \{ \leq, <, =, \neq, \} \), and \( e_1 \) and \( e_2 \) are expressions. A state satisfies a conjunction condition \( s \models k_1 \wedge k_2 \) iff \( s \models k_1 \) and \( s \models k_2 \).

An operator \( o = (\text{pre} \to \text{eff}) \) is applicable in \( s \) iff \( s \models \text{pre} \). The successor state \( \text{app}_o(s) = s' \) resulting from an application of \( o \) is defined as follows, where \( \text{eff} = \{ \text{eff}_1, \ldots, \text{eff}_n \} \): if \( \text{eff}_i \) is a numeric effect \( v_n = e \) with \( o = \{ +, -, \times, \div \} \), then \( s'(v_n) = s(v_n) \circ s(e) \). If \( \text{eff}_i \) is a propositional effect \( v_p = e \), then \( s'(v_p) = s(v_p) \). Finally, if a variable \( v \) does not occur in any effect, then \( s'(v) = s(v) \).

A plan \( \pi \) is a sequence of actions that leads from \( I \) to a state satisfying \( G \) such that each action is applicable in the state that follows by executing the plan up to that action.

We intend to relax numeric planning with the help of intervals. The next section establishes the foundations of interval arithmetic.

**Interval Arithmetic**

Interval arithmetic uses an upper and a lower bound to enclose the actual value of a number. Closed intervals \( [x, \bar{x}] = \{ q \in \mathbb{Q} \mid x \leq q \leq \bar{x} \} \) contain all rational numbers (or \( \pm \infty \)) from \( x \) to \( \bar{x} \). Throughout this paper we refer to the lower bound of an interval \( x \) by \( x \) and to the upper bound by \( \bar{x} \). The set \( I_x = \{ [x, \bar{x}] \mid x \leq \bar{x} \} \) contains all closed intervals. Numbers \( q \) can be transformed into a degenerate interval \( [q, q] \). The basic arithmetic operations in interval arithmetic are given as:

- **addition**: \([x, \bar{x}] + [y, \bar{y}] = [x+y, \bar{x}+\bar{y}] \)
- **subtraction**: \([x, \bar{x}] - [y, \bar{y}] = [x-y, \bar{x}-\bar{y}] \)
- **multiplication**: \([x, \bar{x}] \times [y, \bar{y}] = [\min(x,y), \max(x,y)] \times [\min(y), \max(y)] \)
- **division**: \([x, \bar{x}] \div [y, \bar{y}] = [\min(x/y, \bar{x}/\bar{y}), \max(x/y, \bar{x}/\bar{y})] \)

if \( 0 \notin [y, \bar{y}] \). Otherwise, at least one of the bounds diverges to \( \pm \infty \). We do not explicate all cases of \( x, \bar{x}, y, \bar{y} \) being positive, negative or zero which determine which of the bounds diverge and refer the interested reader to the literature (Moore, Kearfott, and Cloud 2009).

Analogously we define open bounded intervals \( (x, \bar{x}) = \{ q \in \mathbb{Q} \mid x < q \leq \bar{x} \} \) and the set of open intervals \( I_o = \{ (x, \bar{x}) \mid x < \bar{x} \} \), as well as half open intervals \( [x, \bar{x}) = \{ q \in \mathbb{Q} \mid x \leq q < \bar{x} \} \) and \( (x, \bar{x}] = \{ q \in \mathbb{Q} \mid x < q \leq \bar{x} \} \) and the respective sets \( I_{oc} = \{ (x, \bar{x}) \mid x < \bar{x} \} \) and \( I_{oc} = \{ (x, \bar{x}) \mid x < \bar{x} \} \). Finally the set of mixed bounded intervals is given as \( I_m = I_o \cup I_o \cup I_{oc} \cup I_{oc} \). Open and mixed bounded intervals follow the same arithmetic rules as closed intervals. Whenever open and closed bounds contribute to the new interval bound, the bound is open.

**Example 1.** The product \( (-2, 3) \times [-4, 2) \) is the interval \([-12, 8) \). The lower bound is the result of \( 3 \times -4 \) and the resulting bound is closed because both contributing bounds are closed. The new upper bounds is computed by \(-2 \times -4 \) and the open bound of the left interval determines the “openness” of the resulting bound.

**Definition 1.** Let \( x, y \in I_m \) be intervals. The convex union \( u = x \cup y \) is the interval with \( u = \min(x, y) \) and \( \pi = \max(x, y) \). Whether the bounds of \( u \) are open or closed depends on whether those of \( x \) and \( y \) are open or closed.

Definition 1 implicitly adds all values between the intervals to the resulting interval if \( x \cap y = \emptyset \).

**Complexity**

Unlike classical planning, which is PSPACE-complete (Bylander 1994), numeric planning is undecidable (Helmer 2002). Even though completeness of numeric planning can therefore not be achieved in general, numeric planners can find plans or an assurance that the problem is unsolvable for many practical problems. Moreover, we will prove in the following section that the relaxed numeric plan existence problem is decidable in polynomial time for acyclic dependency tasks, in tasks in which the expressions of numeric effects do not depend on the variable they alter.

**Delete Relaxation**

In this section we discuss natural extensions of delete relaxation to planning with numeric variables.

**Motivation**

As planning is hard, it is beneficial to consider a simplified problem in order to obtain guidance in the original problem. The delete relaxation of classical planning ignores delete effects, effects that set the truth value of a proposition to false. As action preconditions and the goal condition require propositions to evaluate to true, delete effects complicate plan search. Finding a relaxed plan on the other hand is possible in polynomial time because relaxed actions do not have delete effects and therefore each action has to be applied at most once. Plans for the original problem are also plans for...
the set of possible values of \( x \) and \( o \) not decidable as solutions have to be integers and the relax-
ting equations as planning problem results in a task that is

**Proof sketch.**
Theorem 1. The numeric plan existence problem in an enu-
mersion relaxation remains undecidable in

**Example 2.** Consider a numeric planning task with ini-
tial state \( I(x) = 0 \) and operators \( o_1 = \{0 \to \{x = +1\} \} \) and \( o_2 = \{0 \to \{x = \pm 2\} \} \). Denoting by \( x_k, k = 0, \ldots, 3 \)
the set of possible values of \( x \) after \( k \) steps, we have:
\[
x_0 = \{0\}, \quad x_1 = \{0, 1\}, \quad x_2 = \{0, \frac{1}{2}, 1, 2\} \quad \text{and} \quad x_3 = \{0, \frac{1}{4}, \frac{1}{2}, 1, 1\frac{1}{2}, 2, 3\}.
\]
For problems with bounded plan length, the enumeration

**Theorem 1.** The numeric plan existence problem in an enu-
meration relaxation is undecidable.

**Proof sketch.** We can basically adopt the proof for nu-
meric planning by Helmert (2002). Formalizing Diophan-
tine equations as planning problem results in a task that is
not decidable as solutions have to be integers and the relax-

**Enumeration.** The number of values that a variable can
attain after applying a fixed number of actions is finite. An
idea is to store the set of all attained values for each vari-

**Discretization.** In order to restrict the number of possi-
ble values from the enumeration approach, multiple values
can be aggregated into “buckets”, where a representative
approximates all values within. These representatives can be
treated as multi-valued finite domain variables from classi-
SAS\textsuperscript{+}-planning. The state transition has to be defined
in such a way that completeness is preserved – plans for the
real problem have to act as plans in the relaxed problem.
Proper abstractions offer potential for future research.

**Higher values are better.** Another approach is only feasi-
bale on a restricted set of planning tasks. If all preconditions
and goals have the form \( (x > c) \) or \( (x \geq c) \) where \( x \) is a nu-
meric variable and \( c \) a numeric constant, higher values are
always beneficial for a variable. Numeric effects are only
allowed to alter numeric variables by a positive constant,
and therefore, decrease effects are considered harmful. The
Metric-FP planning system uses this type of relaxation and
Hoffmann (2003) shows that a large class of problems can be
compiled into the required linear normal form.

**Interval relaxation.** An interval which encloses all val-
ues that a numeric variable can attain is a memory efficient
method. Algebraic base operations are allowed in FDDL and
supported by interval arithmetic. Therefore, we will focus
on interval relaxation in the following section.

**Interval Relaxation**

In this section we elaborate on interval relaxation for nu-
meric planning tasks. We will discuss the complexity of the
plan existence problem for the presented semantics. We
identify a class of tasks with acyclic dependencies between
variables for which we can generate interval relaxed plans in

The interval relaxation of a numeric planning task differs
only marginally from the original task description on a syn-
tactic level. Propositional variables can now be both true and
false at the same time and numeric variables are mapped to
closed intervals.

**Definition 2.** Let \( \Pi \) be a numeric planning task. The in-

terval delete relaxation \( \Pi^+ = \langle \mathcal{V}_+^p, \mathcal{V}_N^+; o^+, I^+, \mathcal{G}^+ \rangle \) of \( \Pi \) is
a 5-tuple where \( \mathcal{V}_+^p \) are the propositional variables from \( \Pi \)
with the domains replaced by \( \text{dom}(v_p) = \{\text{true}, \text{false}, \text{both}\} \)
and \( \mathcal{V}_N^+ \) are the numeric variables with the domains replaced
by closed intervals \( \text{dom}(v_n) = \mathbb{R}_c \) for all \( v_n \in \mathcal{V}_N^+ \).
The initial state \( I^+ \) is derived from \( I \) by replacing numbers
\( I(v_n) \) with degenerate intervals \( I^+(v_n) = [I(v_n), I(v_n)] \)
and \( I^+(v_p) = I(v_p) \). \( \mathcal{G}^+ \) is the goal condition.

The semantic of \( \Pi^+ \) draws on interval arithmetic. Nu-
meric expressions are defined recursively: let \( e_1 \) and \( e_2 \)
be numeric expressions. The interpretation of a constant
expression is \( s^+(c) = [c, c] \) and compound expressions
are interpreted as \( s^+(e_1 \circ e_2) = s^+(e_1) \circ s^+(e_2) \) for
\( \circ \in \{+, -, \times, \div\} \) where “\( o \)” now operates on intervals.
For (goal and operator) conditions, the relaxed semantic is
defined as follows: for \( v_p \in \mathcal{V}_+^p \) be a propositional variable,
then \( s^+(v_p) \) is interpreted as \( [\text{true}, \text{both}] \).
For numeric constraints let \( e_1 \) and \( e_2 \) be numeric expressions,
and \( \circ \in \{<, \leq, =, \neq\} \) a comparison operator. Then
\( s^+(e_1 \circ e_2) \) if \( \circ \in \{<, \leq, =\} \) and \( s^+(e_1 \circ e_2) \) if \( \circ \in \{\neq\} \).
This implies that two intervals can be “greater” and “less”
than each other at the same time.

The semantic of numeric effects \( v_n \circ e \) is relaxed twice:
\( v_n \) keeps its old value and gains all values up to
the new value which is an interval in the relaxation.
The state \( \text{app}^+(s^+) = s^+ \) resulting from an applica-
tion of \( o \) with effect \( e \in \{e_1, \ldots, e_n\} \) is then
\( s^+(v_n) = s^+(v_n) \cup (s^+(v_n) \circ s^+(e)) \) if \( e \) is a numeric ef-
fect. As we use the convex union from Definition 1, \( s^+(v_n) \)
contains all values between the old value of \( v_n \) and the eval-
uated expression \( s^+(v_n) \circ s^+(e) \). For propositional ef-
facts, \( s^+(v_p) = \text{both} \) if the effect changes the truth value
\( v_p \) of \( v_p \), and \( s^+(v_p) = s^+(v_p) \) otherwise.
Again, \( s^+(v) = s^+(v) \) if \( v \) occurs in no effect.

**Example 3.** Applying \( o = (0 \to \{x = e\}) \) in a state map-
ing \( \times \mapsto [8, 10] \) and \( e \mapsto [-\frac{1}{2}, \frac{1}{2}] \) leads to a state
\( s'(x) = [8, 10] \cup [(8, 10) \times [-\frac{1}{2}, \frac{1}{2}]] = [8, 10] \cup [-5, 5] = [-5, 10] \).
Interval Relaxation Complexity

For the classical relaxed planning problem, a relaxed plan can be found by applying all applicable operators in parallel until a fix-point is reached. As no effect can destroy a condition in the relaxed task, the number of operators in the planning task restricts the required number of iterations until a fix-point is reached. The task is solvable if the goal condition holds in the resulting state. A serialized plan can be obtained by ordering actions from the same parallel layer arbitrarily.

We employ a similar method for interval relaxed numeric planning. We have to approach the challenge that numeric operators can have to be applied arbitrarily often. An idea is to transform the planning task into a semi-symbolic representation which captures repeated application of operators with numeric effects. We define interval relaxed and repetition relaxed planning tasks which we refer to as repetition relaxed for short. In repetition relaxed planning tasks we simulate the behavior of applying numeric effects arbitrarily often independently. As we will see later, the independence assumption is not justified for numeric effects \( v_o = e \) where the expression of the assignment \( e \) depends on the affected variable \( v_o \). We show that an adaptation of the algorithm from classical relaxed planning can be used to find plans for repetition relaxed planning tasks with acyclic dependencies, where the variables in \( e \) do not depend on \( v_o \).

Repetition relaxed planning tasks use mixed bounded intervals, intervals whose bounds can either be open or closed, to capture the attainable values of a numeric variable. We are interested in the behavior of numeric effects in the limit. We use different fonts to distinguish a variable and its value \( x \) or \( e \). \( x \) can contract or expand depending on \( e \) which can be an interval relaxed planning task.

\[ \text{Definition 3. Let } \Pi^+ \text{ be an interval relaxed planning task. An (interval and) repetition relaxed planning task of } \Pi^+ \text{ is a } 5\text{-tuple } \Pi^+ = (V_p^+, V_N^+, O^#, I^+, G^#) \text{ with propositional variables } V_p^+, \text{ the domains of numeric variables } \text{dom}(v_o) = I_m \text{ for } v_o \in V_N^+ \text{ are extended to mixed-bounded intervals. The initial state } I^+(v_o) = I^+(v_o) \text{ assigns the same truth value from } I^+ \text{ to each propositional variable } v_o \text{ and each numeric variable } v_o \text{ is initialized to the same closed degenerate interval } I^+(v_o) = I^+(v_o). \]

Again, the relaxation does not change much on a syntactical level. The main difference lies in the semantic of numeric effects. The semantic of numeric expressions can be transferred directly from the interval relaxation as interval arithmetic operations are also defined for mixed bounded intervals. The interpretation of a numeric expression is given as \( s^# (e_1 \circ e_2) = s^# (e_1) \circ s^# (e_2) \) for expressions \( e_1 \) and \( e_2 \) and \( \circ \in \{+, -, \times, \div \} \). The semantic of conditions is again \( s^# (v_o) \text{ iff } s^# (v_o) \in \{ \text{true, both} \} \) for propositions \( v_o \in V^#_p \). For numeric constraints \( e_1 \cup e_2 \) where \( e_1 \) and \( e_2 \) are expressions and comparison operator \( \cup \in \{ <, \leq, =, \neq \} \), \( s^# \text{ iff } \exists y_1 \in s^#(e_1), \exists y_2 \in s^#(e_2) \) with \( y_1 \cup y_2 \).

The semantic of numeric effects captures the repeated application of actions. We first define the repetition relaxed semantic of \( x = e \) for intervals \( x \) and \( e \) with \( x = e \in \{ :=, =+, =-, =\times, =\div \} \). Let \( x_o = x \) and \( x_{i+1} = x_o \cup (x_o \circ e) \) for \( i \geq 0 \) where \( (x : e) \) is defined as \( e \) for assign effects. Let \( \text{succ}_c(x, e) = \bigcup_{\geq 0} x_i \). We are interested in the result of applying an operator arbitrarily often individually for each effect, where the interval \( e \) is fixed even if the expression \( e \) depends on \( x \). As \( x_{i+1} = x \) by definition of the convex union and because all \( x_i \) are convex, the resulting set \( \text{succ}_c(x, e) \) is an interval. However, open bounded intervals can be generated by the limit value consideration. The state \( \text{app}(s^#(v)) = s^#(v) \) resulting from an application of \( o \) with effect \( e = \{ e_1, \ldots, e_n \} \) is then again \( s^#(v) = s^#(v) \) if \( v \) occurs in no effect, \( s^#(v) = \text{both} \) if \( e \neq \emptyset \) and \( s^#(v) \) is a propositional effect which changes the truth value of \( v_o \) and \( s^#(v) = s^#(v) \) otherwise. For numeric effects \( e = \{ v_o = e \} \), \( s^#(v_o) = \text{succ}_c(s^#(v_o), e) \).

Fixing expressions \( e \) of numeric effects \( v_o = e \) to the interval \( e \) they evaluate to in the previous state is beneficial to compute the successor, as changes in the assignment (which can be an arbitrary arithmetic expression) do not have to be considered immediately. The repetition relaxation \( \Pi^# \) of a planning task relaxes \( \Pi^+ \) further and plans for \( \Pi^# \) are still plans for \( \Pi^+ \). The reason is that each operator application can only extend the interval of affected numeric variables more than before. Evaluating the expression in the successor state \( s^#(e) \) only extend the interval \( s^#(e) \).

We want to use the fix-point algorithm which applies all operators of a planning task in parallel until a fix-point is reached to find a repetition relaxed plan. The successors \( \text{succ}_c(x, e) \) of numeric effects are defined by the limit \( \bigcup_{\geq 0} x_i \), and we are interested in determining the result of such an effect in constant time. The result only depends on which of up to 21 symbolic behavior classes are covered by \( x \) and \( e \). The seven behavior classes for \( e \) are \( B_c = \{ (-\infty, -1), (-1, 0), [0, 1), (1, \infty) \} \), and for \( x \) they are \( B_x = \{ (-\infty, 0), [0, 1) \} \). We decompose \( e \) and \( x \) into the hit behavior classes where \( e \cap e \neq \emptyset \) for a behavior class \( e \in B_c \) and \( x \cap x \neq \emptyset \) for a behavior class \( x \in B_x \), respectively. Table 1 contains partial behaviors \( T_c(x, e) \) for \( x \cap e \neq \emptyset \) where \( T_c(x, e) \) is only defined if \( x \subseteq \hat{x} \in B_x \) and \( e \subseteq \tilde{e} \in B_x \). The table entry with column \( \hat{x} \) and row \( \tilde{e} \) in the table with the corresponding \( = \) operator. We use "indeterminate" parentheses \( (\bar{\hat{\tilde{x}}}) \) to denote intervals whose openness is determined by the terms
contributing to it. For assignment effects := we do not need a table as the behavior is equal for all classes, and \( T(x, e) = \{ \min(x, e), \max(x, T) \} \).

<table>
<thead>
<tr>
<th>( + )</th>
<th>( (\infty, 0) )</th>
<th>( {0} )</th>
<th>( (0, \infty) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>( (\infty, \infty) )</td>
<td>( (\infty, T) )</td>
<td>( (x, \infty) )</td>
</tr>
</tbody>
</table>

Table 1: Partial behaviors for numeric effects

Proof for division, \( x \subseteq \hat{x} = (0, \infty) \) and \( e \subseteq \hat{e} = (0, 1) \):
We have to show that \( \text{succ}_x(x, e) = (0, T) \).

“\( \subseteq \)”: In order to prove \( \text{succ}_x(x, e) \subseteq (0, T) \), we show that for every element \( q \in \text{succ}_x(x, e) = \bigcup_{i=0}^{\infty} x_i \) there exists an index \( k \in \mathbb{N} \) with \( q \in x_k = \{x_{k+1}, \ldots\} \) and \( x_k \subseteq (0, T) \). We prove this subset relation separately for each bound of \( x_k \).

**Lower bound:** We show \( x_k > 0 \) for all \( k \in \mathbb{N} \) by induction. The base case \( x_0 > 0 \) holds as \( x_0 = x \subseteq (0, \infty) \). Inductively \( x_{i+1} = x_i \cup (x_i \times e) = \min(x_i, x_i \times e) \) is both positive, the result is positive as well. The minimum is obtained for \( x_i \times e \) because \( e \) is a contraction. Thus, for all \( k \) it holds that \( x_k > 0 \).

**Upper bound:** Again, we show \( \overline{x_k} \subseteq \overline{T} \) for all \( k \in \mathbb{N} \) by induction. The base case holds because \( \overline{x_0} = \overline{x} \) with interval open/closed as for \( x \). The upper bound does not change in the inductive step and we have \( \overline{x_{i+1}} = \overline{x_i} \) because \( \overline{x_{i+1}} = \overline{x_0} \) by definition of the convex union and \( x_{i+1} \subseteq \overline{x_i} \) because \( \overline{x_{i+1}} = \overline{x_i} \). Thus, for every \( q \) it holds that \( q \subseteq \overline{x_i} \). Together with the lower bound \( 0 < q \) we can conclude that \( q \in (0, T) \).

“\( \supseteq \)”: Now we have to show the converse direction \( \text{succ}_x(x, e) \supseteq (0, T) \). Let \( q \in (0, T) \). We have to show that \( q \in \text{succ}_x(x, e) \). As \( \overline{x_k} = \overline{\overline{x}} \) for all \( k \in \mathbb{N} \) we only have to show that there exists a \( k \in \mathbb{N} \) with \( q \in x_k = \{x_{k+1}, \ldots\} \) because \( x_{i+1} \supseteq x_i \) and therefore \( q \in \text{ succ}_x(x, e) = \bigcup_{i=0}^{\infty} x_i \). Such a \( k \) exists because to obtain \( x \times e^k < q \) respectively \( e^k < q \). Building the logarithm alters the inequality because \( e < 0 \): \( \log_e(e^k) > \log_e(q \div x) \). Therefore, \( q \in \text{ succ}_x(x, e) \).

**Proof for division, \( x \subseteq \hat{x} = (0, \infty) \) and \( e \subseteq \hat{e} = (0, 1) \):** We have to show that \( \text{ succ}_x(x, e) = (0, T) \).

“\( \subseteq \)”: We prove \( \text{ succ}_x(x, e) \subseteq (0, T) \) again by showing that for every \( q \in \text{ succ}_x(x, e) = \bigcup_{i=0}^{\infty} x_i \) there exists an index \( k \in \mathbb{N} \) with \( q \in x_k = \{x_{k+1}, \ldots\} \). We show inductively that \( x_k \subseteq (0, T) \) for all \( k \in \mathbb{N} \).

**Base case:** For \( q \in x_0 = (0, T) \) with bounds open/closed as in \( x \), it is easy to show that also \( q \in (0, T) \) because from \( x \subseteq (0, \infty) \) and \( e \subseteq (0, 1) \) we know that \( x, T, e \) and \( T \) are all negative and therefore \( x \times T > 0 \). By the upper bound on the right hand side is always greater than the upper bound of \( x_0 \) and we have \( x = x_0 \leq q < x_0 < T \).

**Inductive step, lower bound:** We have to prove that the lower bound \( x_{i+1} \geq x \), with the induction hypothesis that \( x_k \geq x \) holds. The new bound \( x_{i+1} = x_i \cup (x_i \times e) \) is attainted at \( x_i \) because \( x_0 < e < T < 0 < x_i \) and \( x_i \times e \) is positive and obviously greater than \( x_i \). If the upper bound \( x_i \) is also negative, the minimum for \( x_{i+1} \) is clearly attained at \( x_i \) but even if \( x_i \) is positive, it is bounded by \( 0 \leq x_i \leq x_0 < T \). Because the division by \( e \subseteq (0, 1) \) is a contraction, the highest absolute value is attained by dividing by \( T \) but still \( x_{i+1} \geq x_i \). Therefore, the minimum of \( x_{i+1} \) is \( x_i \). The lower bound remains open/closed for all \( k \) and \( q \geq x_{i+1} = x_i = x_0 = x \).

**Inductive step, upper bound:** We now have to show that \( x_{i+1} \leq \frac{x}{1+e} \). The new bound \( x_{i+1} \) is computed as \( \overline{x_i} = \max(x_i, x_i \times e, x_i \times e) \). The maximum is attained at \( x_i \times e \) (and at \( x_0 \) if they are equal). The reasoning is as follows: all elements of \( e \) are negative and if \( x_i \) and \( \overline{x_i} \) are both negative, \( x_i \) has the higher absolute value. If \( \overline{x_i} \) is positive, the division by a negative number will not contribute to a higher upper bound. As \( e < T < 1 \) is a contraction, the highest value is achieved for \( x \div T \) with bounds closed if the bounds corresponding to \( x \) and to \( T \) are both closed, and open otherwise. With \( x_i > x \) by induction hypothesis, we can therefore conclude...
that $x_{i+1} \leq x_i \leq x_{i+1}$. Therefore, $q \leq x_{i+1} \leq x_i$. Together with the lower bound $x \leq x_{i+1} \leq q$ we can conclude that $q \in \{x, x_{i+1}\}$.

“$\supseteq$”: We have to show that $\text{succ}_x(x, e) \supseteq \{x, x_{i+1}\}$. Let $q \in \{x, x_{i+1}\}$. We have to show that it then also follows that $q \in \text{succ}_x(x, e)$. As $x_k = x$ for all $k \in \mathbb{N}$ we only have to show that there exists $a \in \mathbb{N}$ with $q \in x_a = (x_k, x_\infty)$ because $x_{i+1} \subseteq x_k$ and therefore $q \in \text{succ}_x(x, e) = \bigcup_{i=0}^{\infty} x_i$. The maximum is obtained after $k = 1$ steps because the maximum to compute $x_{i+1} = x_i \leq x$ only depends on the lower bound $x_i$ which equals $x$ for all $k \geq 0$.

With such a decomposition, numeric effects can now be computed in constant time. Unfortunately, the union of the partial behaviors of an effect does not equal the semantic of a successor.

**Hypothesis 1.** The successor $\text{succ}_x(x, e)$ of an effect $x = e$ is the union of the successors obtained by decomposition into the effect into behavior classes, i.e. $\bigcup_{q \in B_x, \bar{e}\in B} \text{succ}_x(x \cap \bar{x}, e \cap \bar{e}) = \text{succ}_x(x, e)$ where $\text{succ}_x(\emptyset, e) = \text{succ}_x(x, 0) = \emptyset$.

Hypothesis 1 does not hold in general, as the following example illustrates. The successor can grow into behavior classes which were not covered by the decomposition:

**Example 4.** Let $o = \langle \emptyset \rightarrow \{x \times e = \} \rangle$ have a numeric effect on $x$ in a state where $x = [1, 4]$ and $e = [-\frac{1}{2}, 2]$. The successor $\text{succ}_x(x, e)$ is $(-\infty, \infty)$. However, the partial behaviors of the decomposition are $\text{succ}_x(x, [-\frac{1}{2}, 0]) = [-2, 4]$, $\text{succ}_x(x, [0, 0]) = [0, 4]$, $\text{succ}_x(x, [0, 1]) = (0, 4]$, $\text{succ}_x(x, [1, 1]) = [1, 4]$ and $\text{succ}_x(x, [1, 2]) = (1, \infty)$. With the union $\text{succ}_x(x \cap \bar{x}, e \cap \bar{e}) = [-2, \infty)$ which differs from $\text{succ}_x(x, e) = (-\infty, \infty)$.

However, the number of behavior classes is restricted, and therefore, new behavior classes can only be hit a restricted number of times. The hypothesis can therefore be fixed by including the partial behaviors $\mathcal{T}_x(x, e)$ of the classes attained by $x$ in a nested fix-point iteration: Let $x_0 = x$ and $x_{j+1} = \bigcup_{q \in B_x, \bar{e}\in B} \text{succ}_x(x_j \cap \bar{x}, e \cap \bar{e})$ with $\text{succ}_x(\emptyset, e) = \text{succ}_x(x, 0) = \emptyset$ for $j \geq 0$. Let $\widetilde{\text{succ}}_x(x, e) = \bigcup_{j=0}^{\infty} x_j$. Now, the newly attained behavior classes become part of the decomposition in the next iteration.

**Example 5.** Recall Example 4 starting with $x_0 = x = [1, 4]$ where the successor $\text{succ}_x(x_0 \cap \bar{x}, e \cap \bar{e}) = [-2, \infty)$. Building the decomposition over the newly achieved behavior classes with $x_1 = [-2, \infty)$ and $e = [-\frac{1}{2}, 2]$ contains among others $\text{succ}_x([-2, 0), (1, 2]) = (-\infty, 0)$. The union still contains partial behaviors which set the upper bound to infinity and therefore $\text{succ}_x(x_1 \cap \bar{x}, e \cap \bar{e}) = (-\infty, \infty))$. Now, a fix-point is reached and $\widetilde{\text{succ}}_x(x, e) = \text{succ}_x(x, e)$.

**Lemma 1.** The union of decomposed successors $\widetilde{\text{succ}}_x(x, e)$ converges after at most 21 steps.

**Proof sketch.** The number of behaviors in each class is restricted to $|B_x| = 3$ and $|B_x| = 7$. Most partial behaviors $\mathcal{T}_x(x, e)$ either set a new bound to a certain value (0 or $\pm \infty$), or leave a bound of $x$ unchanged. The only unsafe cases are multiplications or divisions of a bound with $-1$ or $e$. However, none of these cases is problematic because $e$ is fixed: $\mathcal{T}_x(x, e)$ with $x \leq x \leq (-\infty, 0)$ and $e \leq e = (-1, 0)$ sets a new upper bound $x \times e \geq 0$. However, for all classes $\mathcal{T}_x(x, e)$ with $x \leq x \leq (0, \infty)$, the upper bound is either set to $\infty$ or it remains the same. Therefore no problematic interactions occur. The same reasoning holds for $\mathcal{T}_x(x, e)$ with $e \leq e = (-\infty, -1)$.

The feasibility of a decomposition can therefore be reformulated to the following Theorem:

**Theorem 3.** The successor $\text{succ}_x(x, e)$ of an effect $x = e$ is the fix-point of the convex union of the successors obtained by decomposition of the effect into behavior classes, i.e. $\text{succ}_x(x, e) = \widetilde{\text{succ}}_x(x, e)$.

**Proof sketch.** It should be evident that $\text{succ}_x(x, e) \subseteq \text{succ}_x(x, e)$. In the first iteration of $\text{succ}_x(x, e)$ all partial behaviors $\text{succ}_x(x \cap \bar{x}, e \cap \bar{e})$ are operations on subsets of $x$ and $e$. As interval arithmetic is well defined, an arithmetic operation on a interval $x$ will therefore always subsume the interval resulting from the same operation of a sub-interval $x' \subseteq x$. During each iteration of $\text{succ}_x(x, e)$, the decomposition can only grow to behavior classes that were part of $\text{succ}_x(x, e)$ in the first place.

The converse direction $\text{succ}_x(x, e) \supseteq \text{succ}_x(x, e)$ is shown by contradiction. Let $q \in \text{succ}_x(x, e)$ but not in $\text{succ}_x(x, e)$. Both successor functions are defined recursively starting with $x_0 = x$. Therefore $q \notin x_0$, and there has to be a $k \geq 0$ in $\text{succ}_x(x, e)$ with $k_{k+1} = x_k \cap (x_k \cap e)$ so that $x_k \subseteq \text{succ}_x(x, e)$ but $x_{k+1} \not\subseteq \text{succ}_x(x, e)$. After $k$ steps, the bound of the successors extended beyond the decomposition $\text{succ}_x(x, e)$ for the first time. Obviously, the new bound does not originate in $x_k$ but the new interval $x_{k+1}$ is obtained from $(x_k \cap e)$. The resulting interval depends on $x_k, x_k, e, e$ and in case of division also on whether $0 \in e$. Each combination of these extreme bounds is contained in one partial behavior $\mathcal{T}_x(x_k, e)$. If $(x_k \cap e)$ hits a new behavior class or extends the bounds within a behavior class, this is a contradiction to $\text{succ}_x(x, e)$ being a fix-point. If $(x_k \cap e)$ stays within a behavior, this is a contradiction to $\mathcal{T}_x(x_k, e)$ being well defined (Theorem 2). Thus, such a $k$ cannot be found, and therefore, it is impossible for $q \in \text{succ}_x(x, e)$ but not $q \in \text{succ}_x(x, e)$.

With the help of the decomposed successor $\text{succ}_x(x, e)$ we can compute the result of applying an operator $\text{app}_x^\#$ with the repetition relaxed semantic in constant time. This allows us to use the parallel fix-point algorithm from the classical case analogously: apply all applicable operators in parallel until a fix-point is reached. If the algorithm terminates, the plan is indeed a plan.

**Theorem 4.** The parallel fix-point algorithm for repetition relaxed planning is correct, i.e. if the algorithm outputs an alleged plan, it is indeed a plan for $\Pi^\#$.

**Proof.** Operators are only applied if the precondition is fulfilled.

Unfortunately, the algorithm does not necessarily terminate. In the definition of the semantic of a repetition relaxed planning task, we fix the effect $e$ even if it depends on $x$. However, this implicit independence assumption is
not justified. Inspecting the entries in Table 1 reveals critical entries (marked in red) for multiplicative effects which contract $x$ and flip the arithmetic sign at the same time. The same is true for assignment effects where $T_e(x, e) = (\min(x, e), \max(\tau, \tau))$. In these cases, the new value of $x$ can have a different behavior, if $e$ also depends on $x$. As $e$ can change when $x$ changes, the algorithm does not necessarily terminate.

**Example 6.** Let $x = [-1, -1]$ and $o = \langle \emptyset \rightarrow \{x \equiv e\} \rangle$ with $e = -\frac{x + 1}{2}$. The goal is $G = \{x \geq 1\}$.

Applying the operator arbitrarily often according to the repetitive semantic yields the following progression for $k$ operators:

$$
\begin{array}{c|c|c}
  k & x & e \\
  \hline
  0 & [-1, -1] & [0, 0] \\
  1 & [-1, 0, 5] & [-0,5, 0] \\
  2 & [-1, 0, 75] & [-0.75, 0] \\
  3 & [-1, 0, 875] & [-0.9375, 0] \\
  4 & [-1, 0, 9375] & [-0.9875, 0] \\
  \vdots & \vdots & \vdots \\
\end{array}
$$

Obviously, interval $x$ does not only change a restricted number of times, so the fix-point algorithm for interval relaxed numeric planning will not terminate.

If we succeed in directly computing the fix-point to which the intervals converge with a symbolic interval we could continue the fix-point algorithm from here. In Example 6 we could continue if we would set $x = [-1, 1)$ and $e = (-1, 0]$. Unfortunately, the authors did not succeed in finding a general approach to do so (or to prove that such a general approach does not exist). Instead, we will now restrict the problem to planning tasks where the aforementioned problem does not occur. The problem in Example 6 is that $e$ depends on $x$. Thus, we will restrict planning tasks to contain only effects where the assigned expression is independent from the affected variable. We will then show that such planning tasks are solvable in polynomial time.

**Definition 4.** A numeric variable $v_1$ is directly dependent on a numeric variable $v_2$ in task $\Pi$ if there exists an $o \in O$ with a numeric effect $v_1 \equiv e$ so that $e$ contains $v_2$.

Note that a variable can be directly dependent on itself. Also, the definition of direct dependence does not consider operator applicability.

**Definition 5.** A planning task $\Pi$ is an acyclic dependency task, if the direct dependency relation is acyclic.

Theorem 5. The parallel fix-point algorithm for repetition relaxed planning terminates for acyclic dependency tasks.

**Proof.** As the planning task has acyclic dependencies, the direct dependency relation induces a topology. Let a phase of the algorithm be a sequence of parallel operator applications, where no new operator becomes applicable. During each phase, we consider numeric effects in topological order concerning the dependency graph. Let $V_{N}^{#1} \subseteq V_{N}^{#}$ be the variables in dependency layer $l$. We iterate over the layers $k \geq 0$ of the topology assuming that a fix-point is reached for all variables $V_{N}^{#k}$ of $V_{N}^{#k+1}$ only depend on variables $V_{N}^{#l}$ with $0 \leq l \leq k$ or on constants. A fix-point is reached for all those variables by induction hypothesis. Inductively, we can assume that the expressions of numeric effects which alter the variables of layer $V_{N}^{#k+1}$ are fixed. Therefore, the successor $\text{succ}_{o}(x, e)$ of an effect $(x := e)$ with $x = s_{\text{eff}}(x)$ and $e = s_{\text{eff}}(e)$ does not change the variable more than once (or more than 21 times, if we also consider the intermediate variable updates of the nested fix-point iteration from Lemma 1).

The number of phases is restricted, too, with the same argument as for the fix-point algorithm in the classical case. No precondition can be invalidated once it holds, and during each phase at least one operator which was not applicable before must become applicable. The number of phases is therefore restricted to the number of operators in the planning task.

**Theorem 6.** The fix-point algorithm for repetition relaxed planning is complete for acyclic dependency tasks.

**Proof.** We prove completeness by contradiction and show that it is impossible that the algorithm terminates and reports unsolvable although a plan exists. Now assume there is a plan, but the algorithm terminates and reports unsolvable. Therefore, a satisfiable condition must have been unsatisfied. For propositional conditions, this is impossible, as $s_{\text{eff}}(v_p) \vdash v_p$ if $v_p \in \{true, both\}$ and no effect can set a propositional variable to false. Additionally, all operators are applied as soon as they are applicable. Thus, without loss of generality, a satisfiable numeric constraint was not achieved by the algorithm. This implies that a numeric effect $(v_n \equiv e)$ would have been able to assign a value to a variable which was not reached by our algorithm. Therefore, the successor defined by the semantic $\text{succ}_{o}(s_{\text{eff}}(v_n), s_{\text{eff}}(e))$ has to be different from the successor computed by the algorithm $\text{succ}_{o}(s_{\text{eff}}(v_n), s_{\text{eff}}(e))$ which is impossible for numeric tasks with Theorem 3, a contradiction.

Until now we have an algorithm which can compute parallel plans for repetition relaxed planning tasks in polynomial time for acyclic dependency tasks. As intervals can only grow by applying an operator, the plan can be serialized by applying parallel operators from the same layer in an arbitrary order. Beneficial effects may make the application of some operators unnecessary, but it cannot harm conditions.

We are interested in plans for the interval relaxation without the symbolic description of numeric variables. We will now show that we can derive interval relaxed plans $\pi^*$ from repetition relaxed plans $\pi^+$.

Theorem 7. Plans for the repetition relaxation correspond to plans for the interval relaxation.

**Proof sketch.** A serialization of a repetition relaxed plan $\pi^+ = (o_1, o_2, \ldots, o_n)$ where $0 < i < n$ are the operators applied in the $i$-th step where the same operator can be applied in multiple steps. We seek to find a repetition constant $k_i$ for each operator in order to satisfy the constraints from interval relaxed planning corresponding to those of the repetition relaxed planning plan. However, repetition relaxed
tasks operate on mixed bounded intervals $\mathbb{I}_n$, whereas interval relaxed tasks are restricted to closed intervals. Thus, we have to explicate the interval bounds as well. The repetition relaxed fix-point algorithm is split into phases, were during each phase the same operators are applicable. Within each phase, the operators are applied in parallel at most 21 times for each variable (if all variables have different topology levels in the dependence graph). In order to determine the repetition constants $k_+$, we look at each constraint $[a, \overline{a}] \bowtie [b, \overline{b}]$. By definition of $\bowtie$, there exist $q_a$ and $q_b$ in the respective intervals so that $q_a \bowtie q_b$. Let $q_a$ and $q_b$ be such numbers which satisfy the constraint $a \bowtie b$ where $a$ and $b$ are intervals which are obtained by evaluating the corresponding expressions $s^\#(e_a)$ and $s^\#(e_b)$. We investigate each expression individually. For each expression we have a target value $q$. For the constraints above the expressions are $e_a$ and $e_b$ and the corresponding target values $q_a$ and $q_b$. Unless the expression is a variable, the target value has to be obtained recursively from the expressions $e_1 \bowtie e_2$.

**Example 7.** Let $x = [0, 1), y = [0, 1)$ and $z = (1, 7, 3]$ be the symbolic values of variables $x$, $y$ and $z$ with a condition $x + y > z$. From $e_x = x + y$ we choose an arbitrary $q_x = 1.9 \in s^\#(e_x)$ an arbitrary $q_y = 1.8 \in s^\#(e_y)$ from within the expression intervals so that the constraint is satisfied. Now we have to recursively find appropriate $q_x$ and $q_y$ in the sub-expressions. A leeway of $2 - 1.9 = 0.1$ can be distributed arbitrarily to the target values of the sub-expressions. We could for example continue with target values 0.95 for $x$ and $y$ each.

We can choose arbitrary target values for the sub-expressions within a leeway of feasible choices. Eventually, all expressions induce target values for the numeric variables. This can induce multiple different target values for each variable where only the most extreme target values have to be considered (an interval including the most extreme target values will also include intermediate target values). All target values originate from a repetition relaxed symbolic state, so they are indeed reachable. In the repetition relaxed plan, each operator which has a numeric effect on a variable with a target value achieved the symbolic value for this variable with a partial behavior from Table 1. For each operator, the constant $k$ is now computed by solving $x \pm k \cdot e = q$ for additive effects and $x \cdot e^k = q$ for multiplicative effects. The $k$ is therefore the same $k$ from the proof of Theorem 2. Each operator then has to be applied $n$ times, where $n$ is the sum over all $k_n$ in the phases of the algorithm, where $k_n$ is the maximum number of applications required for that operator in that phase.

**Theorem 8.** The problem to generate an interval relaxed numeric plan is in $P$ for tasks with acyclic dependencies.

**Proof.** The fix-point algorithm for repetition relaxed planning tasks is correct (Theorem 4) and complete (Theorem 6) and it terminates in polynomial time (Theorem 5). Therefore, generating a repetition relaxed plan $\pi^\#$ is possible in polynomial time. An interval relaxed plan $\pi^+$ can be constructed from $\pi^\#$ (Theorem 7) in polynomial time.

The definition of a relaxation is adequate (Hoffmann 2003) if it is admissible, i.e. any plan $\pi$ for the original task $\Pi$ is also a relaxed plan for $\Pi^+$, if it offers basic informedness, i.e. the empty plan is a plan for $\Pi$ iff it is a plan for $\Pi^+$ and finally the plan existence problem for the relaxation is in $P$.

**Theorem 9.** The interval relaxation is adequate for acyclic dependency tasks.

**Proof.** Admissibility. After each step of the plan $\pi$, if propositional variables of the relaxed state differ from the original state, they assign to both which cannot invalidate any (goal or operator) conditions. The original value of numeric variables is contained in the interval of the relaxed state. As comparison constraints are defined with the relaxed semantic that a constraint holds if it holds for any pair of elements from the two intervals, admissibility follows directly.

Basic informedness. No (goal or operator) conditions are dropped from the task. Relaxed numeric variables are mapped to degenerate intervals which only contain one element. Therefore, conditions in the original task $x \bowtie y$ correspond to interval constraints $[x, x] \bowtie [y, y]$ which are satisfied if they are satisfied in the relaxed task.

Polynomiality. As a corollary to Theorem 8, we can also conclude that interval relaxed numeric plan existence is in $P$ for tasks with acyclic dependencies.

The interval relaxation is admissible and offers basic informedness. For acyclic dependency tasks, the plan existence problem can be decided in polynomial time. Thus, the interval relaxation is adequate.

The proposed relaxation advances the state of the art even though the adequacy of interval relaxation was only shown for a restricted set of tasks. However, the requirement of acyclic dependency for numeric expressions is a strict generalization of expressions $e$ being required to be constant, which is required for other state-of-the-art approaches e.g. (Hoffmann 2003). On the practical side, many interesting planning problems are restricted to constant expressions.

**Conclusion and Future Work**

We presented interval algebra as a means to carry the concept of a delete relaxation from classical to numeric planning. We proved that this relaxation is adequate for acyclic dependency tasks, tasks where the expressions of numeric effects do not depend on the affected variable. The complexity of the approach for arbitrary interval relaxed planning problems remains an open research issue though. It is imaginable that a clever approach can find the fix-point of arbitrary operator application in polynomial time.

In the future, we intend to adapt the most iconic heuristics from classical planning, $h_{max}$, $h_{add}$ and $h_{FF}$ to the interval relaxation framework.

**Acknowledgments**

This work was partly supported by the DFG as part of the SFB/TR 14 AVACS.


