Representing Action Domains with Numeric-Valued Fluents

Esra Erdem[1] and Alfredo Gabaldon[2,3]

[1] Institute of Information Systems,
Vienna University of Technology, Vienna, Austria
[2] National ICT Australia
[3] School of Comp. Sci. and Eng., UNSW, Sydney, Australia

Abstract. We present a general method to formalize action domains with numeric-valued fluents whose values are incremented or decremented by executions of actions, and show how it can be applied to the action description language $\mathcal{C}+$ and to the concurrent situation calculus. This method can handle nonserializable concurrent actions, as well as ramifications on numeric-valued fluents, which are described in terms of some new causal structures, called contribution rules.

1 Introduction

Numeric-valued fluents are used for describing measurable quantities, such as weight, money, memory. In many cases, the values of such fluents are incremented/decremented by the execution of actions, such as adding/removing some weight, depositing/withdrawing some money, or allocating/deallocating memory. How to compute the value of a numeric-valued fluent after a concurrent execution of several such actions, possibly with indirect effects, is the question we study in this paper. We consider true concurrency: actions occur at the same time and may not be serializable (i.e., their effect may not be equivalent to the effect of executing the same actions consecutively in any order). For instance, consider two boats towing a barge upriver by applying forces via cables tied to the barge, where the force applied by either boat is not enough to move the barge against the current of the river; here the concurrent action of two boats applying forces can not be serialized. True concurrency makes the problem more challenging, because actions that are individually executable may not be executable concurrently, e.g., due to conflicting effects, and actions that are individually nonexecutable may be executable concurrently, e.g., due to synergistic effects, like in the example above.

This question is important for real-world applications that involve reasoning tasks, like planning or prediction, related to resource allocation. For instance, allocation of memory storage for use by computer programs is one such application. It is also important for applications that involve modeling the behavior of physical systems. For instance, how water pressure changes at a piston when some water is pumped from above and some force is applied from the bottom is important for modeling the behavior of a hydraulic elevator.

There are several planning systems designed to work in concurrent domains with resources, like [1–3]. However, they consider a simpler concurrency: they either re-

quire the serializability of actions, or that no concurrent action contain two actions, one producing and the other consuming the same resource.

Lee and Lifschitz [4] show, in the action language $\mathcal{C}+$ [5], how to formalize action domains involving additive fluents—numeric-valued fluents on which the effect of a concurrent action is computed by adding the effects of its primitive actions. However, since additive fluents range over finite sets, a concurrent action is executable only if its effect on each additive fluent is in that fluent's range, and it is not easy to handle indirect effects of actions (ramifications) on additive fluents (e.g., an indirect effect of adding too much water into a small container is an increase in the amount of water in a large container, into which the excess water overflows from the small container). Similarly, [6] defines the cumulative direct effects of concurrent actions on additive fluents, in an extension of the action language $\mathcal{A}$ [7]; however, it is not easy to handle ramifications (not only the ones on numeric-valued fluents) in this formalism.

In [8], the authors show, in the concurrent situation calculus [9], how to formalize action domains containing numeric-valued fluents, that do not require serializability of actions, and that take into account ramifications caused by too much increase/decrease of a numeric-valued fluent. However, with this formalization, it is not easy to capture other forms of ramifications (e.g., whenever the amount of water increases in the large container, the force towards the bottom of the container increases).

In this paper, we present a general method to formalize action domains with numeric-valued fluents whose values are incremented/decremented by executions of actions. This method is applicable to both the concurrent situation calculus and the action language $\mathcal{C}+$; and thus can be used with the reasoning systems CCALC and GOLOG. The idea is to compute the total effect of a concurrent action on a numeric-valued fluent, in terms of the direct and the indirect effects of its primitive actions on that fluent, while also taking into account the range restrictions (e.g., the capacity of the small container).

To describe direct effects, like in [4, 8], we introduce new constructs and functions in the original formalisms. To describe ramifications, like in [10–12], we introduce an explicit notion of causality, specific for numeric-valued fluents. We characterize this notion by *contribution rules*, motivated by the equation-like causal structures of [13, 14, 8]. With contribution rules, both forms of ramifications above can be handled. The idea of introducing these new constructs is to be able to represent effects of actions on numeric-valued fluents concisely. Semantically these constructs are treated as "macros" on top of the original formalisms; like the constructs introduced in [4] and in [8], they are compiled into causal laws or formulas in the original formalisms.

The paper consists of three parts. The first two parts describe how action domains with numeric-valued fluents can be formalized in the action language $\mathcal{C}+$ and in the concurrent situation calculus, using the new constructs; the semantics of these constructs is defined by showing how to treat them as abbreviations in the original formalims. The third part includes a comparison of these two formalizations, and a discussion of related work. Appendix contains the proofs, and the CCALC and GOLOG files describing our running example.

2 The Action Description Language $\mathcal{C}$+

We briefly describe the action description language $\mathcal{C}$+ of [5] below.

We start with a set of symbols, called *constants*; each constant e is associated with a nonempty finite set *Dom*(e) of symbols. The constants are divided into two: *fluent constants* and *action constants*. An *atom* is an expression of the form $e = v$ where e is a constant and $v \in$ *Dom*(e). We use e (respectively, $\neg e$) instead of $e =$ *true* (respectively, $e =$ *false*). A *formula* is a propositional combination of atoms. A *fluent formula* (respectively, an *action formula*) is a formula such that all constants appearing in it are fluent (respectively, action) constants.

The action description language $\mathcal{C}$+ consists of three kinds of expressions (called *causal laws*): *static laws* of the form

$$\textbf{caused } \phi \textbf{ if } \psi \tag{1}$$

where ϕ and ψ are fluent formulas; *action dynamic laws* of the form (1) where ϕ is an action formula and ψ is a formula; and *fluent dynamic laws* of the form

$$\textbf{caused } \phi \textbf{ if } \psi \textbf{ after } \rho \tag{2}$$

where ϕ and ψ are fluent formulas, and ρ is a formula. An *action description* is a set causal laws.

The meaning of an action description can be represented by a transition diagram—a directed graph whose every vertex denotes a state and every edge denotes transitions due to action occurrences. (See [5] for a more precise definition and an example.)

In the next sections we use the following three constructs, each standing for some causal laws as shown in [5]. If a is a boolean action constant, then we can describe that a is not executable if ϕ, by the expression

$$\textbf{nonexecutable } a \textbf{ if } \phi;$$

we can express that a is exogenous (i.e., the causes of the occurrence or nonoccurrence of the action c is not given in the action domain description) by

$$\textbf{exogenous } a;$$

We can describe that the value of a fluent constant f is v by default, with the expression

$$\textbf{default } f = v.$$

3 Describing Additive Fluents in the Action Language $\mathcal{C}$+

To formalize action domains with additive fluents, we extend the action description language $\mathcal{C}$+, similar to [4].

Additive fluents According to this extension, some numeric-valued fluent constants can be designated as *additive*. Each additive fluent constant has a finite set of numbers as its domain. As in [4], we understand numbers as symbols for elements of any set with an associative and commutative operation $+$ that has a neutral element 0; in particular, we consider the additive group of integers (since this case can be implemented for CCALC). We suppose that the domain of each additive fluent constant f is specified as a range $[L_f, U_f]$, so that, at any state, $L_f \leq f \leq U_f$. We suppose that heads of causal laws do not contain any additive fluent constants.

Direct effects of actions Direct effects of a boolean action constant a on an additive fluent f are expressed by *increment laws* of [4], expressions of the form

$$a \textbf{ increments } f \textbf{ by } n \textbf{ if } \psi \tag{3}$$

where n is an integer and ψ is a fluent formula. We drop the '**if** ψ' part if $\psi \equiv \top$; we call f the *head* of the causal law. Intuitively, an increment law of form (3) expresses that, if ψ holds, the direct contribution of the action a to the value of the additive fluent f is n. The idea is then, to compute the cumulative direct contribution of concurrently executed primitive actions to the value of an additive fluent f, denoted *DContr*(f), by adding the direct contributions of those primitive actions to f. Translation of these laws into causal laws is different from that of [4] (see the definition of *DContr* in the next section).

Preconditions of actions We describe preconditions of actions with the **nonexecutable** construct of [5]. For instance, the expression

$$\textbf{nonexecutable } \mathit{Move}(A, B) \textbf{ if } \neg \mathit{Clear}(B)$$

describes that moving Block A onto Block B is not possible if B is not clear.

Ramifications on additive fluents Ramifications on an additive fluent f are described by *contribution rules*, expressions of the form:

$$f \xleftarrow{\oplus} \mathcal{E}(h) \tag{4}$$

where h is one of the additive fluents that f depends on, $\mathcal{E}$ is a numeric-valued function, and $\oplus$ is an element of $\{+, -, ++, +-, -+, --\}$; we call f the *head* of the rule. These rules allow us to describe both kinds of ramifications mentioned in the introduction. The first kind of ramifications is expressed with $\oplus = +$ or $\oplus = -$. The meaning of a rule of form (4) with $\oplus = +$ (respectively, with $\oplus = -$) can be described as follows: whenever the sum of the direct and indirect contributions of a concurrent action to h, when added to h, exceeds the upper bound U_h (respectively, goes beyond its lower bound L_h), that action indirectly contributes to f by the amount $\mathcal{E}(\mathit{DContr}(h) + \mathit{IContr}(h) - \mathit{TContr}(h))$, where *IContr*$(h)$ denotes the indirect contribution of a concurrent action to h, and *TContr*(h) denotes the total contribution of a concurrent action to h respecting the range restriction $[L_h, U_h]$. Intuitively, *DContr*(h) + *IContr*(h) − *TContr*(h) describes the excess amount being contributed to h.

The other form of ramifications is expressed with $\oplus \in \{++, +-, -+, --\}$. A rule of form (4) with $\oplus = ++$ (respectively, with $\oplus = +-$) expresses that whenever there is an increase (respectively, decrease) n in the value of h, i.e., *TContr*$(h) = n$, the value of f increases (respectively, decreases) by $\mathcal{E}(n)$; the rules with $\oplus \in \{-+, --\}$ are similar, but they specify a decrease in the value of f. This form of ramification, unlike the one above, is not due to the range restrictions imposed on the values of fluents, although these restrictions must be satisfied at all times.

The indirect contribution of an action to an additive fluent f is the sum of the increases/decreases described by the contribution rules with the head f.

Once the direct and indirect contributions of a concurrent action to an additive fluent f are computed, we can compute the total contribution of that action to f as follows. If

f appears on the right hand side of a contribution rule of form (4) with $\oplus = +, -$, then we add *DContr*(f) and *IContr*(f), considering the range restriction $[L_f, U_f]$:

$$\textit{TContr}(f) = \begin{cases} U_f - f & \text{if } \textit{DContr}(f) + \textit{IContr}(f) > U_f - f \\ L_f - f & \text{if } \textit{DContr}(f) + \textit{IContr}(f) < L_f - f \\ \textit{DContr}(f) + \textit{IContr}(f) & \text{otherwise.} \end{cases}$$

Otherwise, we do not need to consider the range restriction, and *TContr*(f) is defined as *DContr*(f)+*IContr*(f).

We consider action domains only where the causal influence among fluents is acyclic. Here is an example.

Example 1. Consider three containers, small, medium, and large, for storing water. The small container is suspended over the medium, and the medium container is suspended over the large so that, when the small (respectively, medium) container is full of water, the water poured into the small (respectively, medium) container overflows into the medium (respectively, large) container. Suppose that there are three taps: one directly above the small container, by which some water can be added to the containers from an external source, one on the small container, by which some water can be released into the medium container, and a third tap on the large container to release water to the exterior. Suppose also that one unit increase (respectively, decrease) of water in the large container increases (respectively, decreases) the amount of force applied downwards to the bottom of the large container by two units. Also assume that some force is exerted upwards at the bottom of the large container, e.g., by a piston, to lift it up.

A formalization of this action domain in the extended $\mathcal{C}$+ is presented in Figure 1. Here the additive fluent constants *Small*, *Medium*, and *Large* describe the amount of water in each container; *Force* describes the force exerted upwards at the bottom of the large container. The boolean action constant *AddS*(n) describes the action of adding n units of water to the small container by opening the tap over it; *ReleaseS*(n) and *ReleaseL*(n) describe the action of releasing n units of water from the small, respectively large, container by opening its tap; and *Exert*(n) represents the action of exerting n amount of force upwards.

Suppose that the range restrictions are specified as follows: $L_{\textit{Small}} = L_{\textit{Medium}} = L_{\textit{Large}} = 0,\ L_{\textit{Force}} = -8, U_{\textit{Small}} = 2,\ U_{\textit{Medium}} = 3,\ U_{\textit{Large}} = 4,\ U_{\textit{Force}} = 8$. If initially *Small* $=$ *Medium* $=$ *Large* $= 1$, *Force* $= -2$, then, after executing the concurrent action $c = \{\textit{AddS}(8), \textit{ReleaseS}(1), \textit{ReleaseL}(2), \textit{Exert}(8)\}$, the values of fluents are computed by CCALC as follows: *Small* $= 2$, *Medium* $= 3$, *Large* $= 4$, *Force* $= 0$.

Indeed, the direct effect of c on *Small* is the sum of the direct contributions of its primitive actions (described by the increment laws with the head *Small*, in Figure 1): *DContr*(*Small*) $= 8 - 1 = 7$. Since there is no contribution rule with the head *Small*, in Figure 1, there is no ramification on it: *IContr*(*Small*) $= 0$. Since *Small* + *DContr*(*Small*) + *IContr*(*Small*) $= 7$ exceeds the capacity of the small container, the total contribution of c to *Small* is just the amount that fills the small container: *TContr*(*Small*) $= U_{\textit{Small}} -$ *Small* $= 2 - 1 = 1$. Then the value of *Small* after the execution of c is 2.

On the other hand, since the function $\mathcal{E}$ in *Medium* $\xleftarrow{+}$ *Small* is the identity function, the indirect contribution of c to *Medium* is the amount of the excess water overflown into the medium container: *DContr*(*Small*) + *IContr*(*Small*) − *TContr*(*Small*) $=$

Notation: n ranges over $\{Min, .., Max\}$ and a ranges over action constants.

Action constants:	Domains:
AddS(n), *ReleaseS*(n), *ReleaseL*(n), *Exert*(n)	Boolean

Additive fluent constants:	Domains:
Small	$\{L_{Small}, .., U_{Small}\}$
Medium	$\{L_{Medium}, .., U_{Medium}\}$
Large	$\{L_{Large}, .., U_{Large}\}$
Force	$\{L_{Force}, .., U_{Force}\}$

Causal laws:

AddS(n) **increments** *Small* **by** n
ReleaseS(n) **increments** *Small* **by** $-n$
ReleaseS(n) **increments** *Medium* **by** n
ReleaseL(n) **increments** *Large* **by** $-n$
Exert(n) **increments** *Force* **by** n

nonexecutable *AddS*(n) **if** *AddS*(n') $(n \neq n')$
nonexecutable *ReleaseS*(n) **if** *ReleaseS*(n') $(n \neq n')$
nonexecutable *ReleaseL*(n) **if** *ReleaseL*(n') $(n \neq n')$
nonexecutable *Exert*(n) **if** *Exert*(n') $(n \neq n')$

exogenous a

Contribution rules:

Medium $\overset{+}{\longleftarrow}$ *Small* *Large* $\overset{+}{\longleftarrow}$ *Medium*
Force $\overset{+-}{\longleftarrow}$ $2 \times$ *Large* *Force* $\overset{-+}{\longleftarrow}$ $2 \times$ *Large*

Fig. 1. Containers domain described in the extended C+.

$7+0-1=6$. Since the direct contribution of c to *Medium* is 1, the total contribution of c to *Medium* is just the amount that fills the medium container: *TContr*(*Medium*) $= 2$. Then, after the execution of c, *Medium* $= 3$.

Similarly, the direct and indirect contributions of c to *Large* can be computed as follows: *DContr*(*Large*) $= -2$, *IContr*(*Large*) $= 5$. Since *Large* does not appear on the right hand side of a contribution rule of form (4) with $\oplus = +, -$, the total contribution of c to *Large* is simply the addition of these two: *TContr*(*Large*) $= 3$. Then the value of *Large* after the execution of c is 4.

Since the total contribution of c to *Large* is 3, and since the function $\mathcal{E}$ in *Force* $\overset{-+}{\longleftarrow}$ $2 \times$ *Large* is $(\lambda x.2 \times x)$, the indirect contribution of c to *Force* is $-(2 \times 3) = -6$. Since the direct contribution of c to *Force* is +8, the total contribution of c to *Force* is 2. Therefore, the value of *Force* after the execution of c is 0.

4 Obtaining an Action Description

To obtain an action description in $\mathcal{C}$+ from a formalization of an action domain like in Figure 1, we translate increment laws, and contribution rules into causal laws as follows.

1. To describe the direct effects of primitive actions, first we introduce new action constants, *Contr*(a, f), of sort integer, where a is an action constant and f is an additive fluent constant; an atom of the form *Contr*$(a, f) = v$ expresses that the action a contributes to f by the amount v. We define *Contr*(a, f) to be 0 by default:

$$\textbf{default } \mathit{Contr}(a, f) = 0.$$

Then we replace every increment law (3) with

$$\textbf{caused } \mathit{Contr}(a, f) = n \textbf{ if } a \wedge \psi.$$

2. To describe the cumulative effects of concurrent actions, we introduce new action constants, $\mathit{DContr}(f)$, $\mathit{IContr}(f)$, $\mathit{TContr}(f)$, of sort integer, where f is an additive fluent constant. Intuitively, an atom of the form $\mathit{DContr}(f) = v$ (respectively, $\mathit{IContr}(f) = v$) expresses that the direct (respectively, indirect) contribution of a concurrent action to f is v. An atom of the form $\mathit{TContr}(f) = v$ expresses that the total contribution of a concurrent action to f is v.

   We define $\mathit{DContr}(f)$ as follows:

$$\textbf{caused } \mathit{DContr}(f) = \textstyle\sum_a v_a \textbf{ if } \bigwedge_a \mathit{Contr}(a, f) = v_a$$

   where $\mathit{Min} \leq \sum_a v_a \leq \mathit{Max}$.

   Let us denote by C the set of all contribution rules. We define $\mathit{IContr}(f)$ to be 0 by default:

$$\textbf{default } \mathit{IContr}(f) = 0.$$

   Then we translate contribution rules in C into the causal laws:

$$\begin{array}{l}
\textbf{caused } \mathit{IContr}(f) = v \textbf{ if } v = \\
\quad \sum_{f \xleftarrow{+} \mathcal{E}(h) \in C} \mathcal{E}(\mathit{IContr}(h) + \mathit{DContr}(h) - \mathit{TContr}(h)) \\
-\sum_{f \xleftarrow{-} \mathcal{E}(h) \in C} \mathcal{E}(\mathit{IContr}(h) + \mathit{DContr}(h) - \mathit{TContr}(h)) \\
+\sum_{f \xleftarrow{++} \mathcal{E}(h) \in C, \mathit{TContr}(h) > 0} \mathcal{E}(\mathit{TContr}(h)) \\
+\sum_{f \xleftarrow{+-} \mathcal{E}(h) \in C, \mathit{TContr}(h) < 0} \mathcal{E}(\mathit{TContr}(h)) \\
-\sum_{f \xleftarrow{-+} \mathcal{E}(h) \in C, \mathit{TContr}(h) > 0} \mathcal{E}(\mathit{TContr}(h)) \\
-\sum_{f \xleftarrow{--} \mathcal{E}(h) \in C, \mathit{TContr}(h) < 0} \mathcal{E}(\mathit{TContr}(h)) \quad (\mathit{Min} \leq v \leq \mathit{Max}).
\end{array}$$

   For instance, with the contribution rules in Figure 1, for *Medium*, we add

$$\begin{array}{l}
\textbf{caused } \mathit{IContr}(\mathit{Medium}) = v \textbf{ if} \\
\quad \mathit{IContr}(\mathit{Small}) + \mathit{DContr}(\mathit{Small}) - \mathit{TContr}(\mathit{Small}) = v \ \ (\mathit{Min} \leq v \leq \mathit{Max}).
\end{array}$$

   If f appears on the right hand side of a contribution rule of form (4), then we define $\mathit{TContr}(f)$ by adding the direct and indirect contributions of actions, respecting the range restriction $[L_f, U_f]$:

$$\begin{array}{l}
\textbf{caused } \mathit{TContr}(f) = v + v' \textbf{ if } \mathit{DContr}(f) = v \wedge \mathit{IContr}(f) = v' \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (L_f \leq v + v' + f \leq U_f) \\
\textbf{caused } \mathit{TContr}(f) = U_f - f \textbf{ if } \mathit{DContr}(f) = v \wedge \mathit{IContr}(f) = v' \ \ (v + v' + f > U_f) \\
\textbf{caused } \mathit{TContr}(f) = L_f - f \textbf{ if } \mathit{DContr}(f) = v \wedge \mathit{IContr}(f) = v' \ \ (v + v' + f < L_f)
\end{array}$$

   such that the values assigned to $\mathit{TContr}(f)$ are in the range $[\mathit{Min}, \mathit{Max}]$. Otherwise, we define $\mathit{TContr}(f)$ simply by adding the direct and indirect contributions of actions, i.e., by the first set of causal laws above.

3. To determine the value of an additive fluent constant f after an execution of a concurrent action, we add

$$\textbf{caused } f = v + v' \textbf{ if } \top \textbf{ after } f = v \wedge \mathit{TContr}(f) = v' \ \ (\mathit{Min} \leq v + v' \leq \mathit{Max}).$$

With the translation above, the meaning of an action description D in the extended $\mathcal{C}+$ can be represented by the transition diagram described by the action description D' obtained from D as described above (see [7] for a definition of a transition diagram). Then a query Q (in a query language, like $\mathcal{R}$ [7]), which describes a planning problem, a prediction problem, etc., is entailed by D if Q is entailed by D'. This allowed us to compute the values of additive fluents in Example 1 using CCALC.

5 The Concurrent Situation Calculus

The concurrent situation calculus is a second-order language with equality. It has four sorts: *action* for primitive actions, *concurrent* for concurrent actions, *situation* for situations, and *object* for objects in the domain. Intuitively, situations are sequences of concurrent actions representing possible evolutions of the world. Concurrent actions are (possibly infinite) sets of primitive actions. In addition to the standard logical symbols and connectives, the language includes:

- variable symbols of each sort;[4]
- a constant S_0 of sort *situation* denoting the initial situation;
- a function symbol *do* : (*concurrent* $\times$ *situation*) $\mapsto$ *situation* to denote the situation that results after the execution of a sequence of actions;
- a predicate symbol $\sqsubset$ *situation* $\times$ *situation*, with $s \sqsubset s'$ meaning that situation s precedes s';
- functions of the form $A(\boldsymbol{x})$ of sort *action* (with arguments $\boldsymbol{x}$ of sort *object*^{n}), which denote primitive actions;
- predicates of the form $F(\boldsymbol{x}, s)$ and functions of the form $f(\boldsymbol{x}, s)$, with the last argument s always being of sort *situation*, which denote *relational fluents* and *functional fluents*, i.e., properties of the world that change as a result of the execution of actions; and
- a predicate symbol *Poss* of sort (*action* $\cup$ *concurrent*) $\times$ *situation*, which is used to describe whether a primitive or concurrent action is possible in a situation.

Sets and reals are not axiomatized; their standard interpretation, including their operations and relations, is considered.

A *basic action theory* is composed of five sets of axioms:

1. Four *foundational axioms* that are domain-independent:

$$\begin{gathered}\mathit{do}(c_1, s_1) = \mathit{do}(c_2, s_2) \supset c_1 = c_2 \wedge s_1 = s_2,\\ (\forall P) P(S_0) \wedge (\forall c, s)[P(s) \supset P(\mathit{do}(c, s))] \supset (\forall s) P(s),\\ \neg s \sqsubset S_0,\\ s \sqsubset \mathit{do}(c, s') \equiv s \sqsubset s' \vee s = s'.\end{gathered}$$

[4] In the situation calculus constructs below, lower-case Roman letters denote variables. We use s, a, c and x, possibly with superscripts and subscripts, for variables of sorts *situation*, *action*, *concurrent*, and *object*. Unless stated otherwise, variables are implicitly universally prenex quantified.

2. For each primitive action $A(\boldsymbol{x})$, an *action precondition axiom* of the form:

$$\mathit{Poss}(A(\boldsymbol{x}), s) \equiv \Pi_A(\boldsymbol{x}, s)$$

where $\Pi_A(\boldsymbol{x}, s)$ is a formula uniform in s.[5] Minimal $Poss$ requirements for concurrent actions are as follows:

$$\mathit{Poss}(a, s) \supset \mathit{Poss}(\{a\}, s), \tag{5}$$

$$\mathit{Poss}(c, s) \supset (\exists a) a \in c \wedge (\forall a)[a \in c \supset \mathit{Poss}(a, s)]. \tag{6}$$

3. For each relational fluent $F(\boldsymbol{x}, s)$, a *successor state axiom* of the form:

$$F(\boldsymbol{x}, \mathit{do}(c, s)) \equiv \gamma_F^+(\boldsymbol{x}, c, s) \vee F(\boldsymbol{x}, s) \wedge \neg\gamma_F^-(\boldsymbol{x}, c, s)$$

where $\gamma_F^+(\boldsymbol{x}, c, s)$ and $\gamma_F^-(\boldsymbol{x}, c, s)$ are formulas uniform in s. Similarly, for each functional fluent $f(\boldsymbol{x}, s)$, a successor state axiom of the form:

$$f(\boldsymbol{x}, \mathit{do}(c, s)) = y \equiv \\ \gamma_f(\boldsymbol{x}, y, c, s) \vee f(\boldsymbol{x}, s) = y \wedge \neg(\exists y')\gamma_f(\boldsymbol{x}, y', c, s).$$

4. Unique names axioms for actions, such as $move(x, y) \neq pickup(x)$.
5. Axioms describing the initial situation of the world: a finite set of sentences uniform in S_0.

6 Describing Additive Fluents in the Concurrent Situation Calculus

To formalize action domains with additive fluents, we extend the concurrent situation calculus, as in [8].

Additive fluents According to this extension, some functional fluents that range over numbers (not necessarily integers) can be designated as *additive*. For each additive fluent f, we understand a given range $[L_f, U_f]$ as follows: in every situation s, $L_f \leq f(s) \leq U_f$.

Direct effects of actions For describing direct effects of actions on additive fluents, we introduce a function $\mathit{contr}_f(\boldsymbol{x}, a, s)$ for each additive fluent f. Intuitively, $\mathit{contr}_f(\boldsymbol{x}, a, s)$ is the amount that the action a contributes to f when executed in situation s. In the following, free variables are implicitly universally quantified. We describe the direct effects of primitive actions on additive fluents by axioms of the form:

$$\kappa_f(\boldsymbol{x}, v, a, s) \supset \mathit{contr}_f(\boldsymbol{x}, a, s) = v \tag{7}$$

where $\kappa_f(\boldsymbol{x}, v, a, s)$ is a first-order formula whose only free variables are $\boldsymbol{x}, v, a, s$, doesn't mention function contr_g for any g, and s is its only term of sort situation. If there is no axiom (7) describing the effect of an action a on an additive fluent f, we assume that the direct contribution of a to f is zero. This assumption allows us to derive, for each function contr_f, a definitional axiom:

$$\mathit{contr}_f(\boldsymbol{x}, a, s) = v \equiv \kappa_f(\boldsymbol{x}, v, a, s) \ \vee v = 0 \wedge \neg(\exists v')\kappa_f(\boldsymbol{x}, v', a, s).$$

[5] A formula uniform in s does not contain any situation term other than s. (See Definition 4.4.1 of [9].)

Notation: n, n', v are object (number) variables, s is a situation variable, a, a' are action variables, and c is a concurrent variable.

Action functions: *addS*(n), *releaseS*(n), *releaseL*(n), *exert*(n).

Additive fluent functions:	Ranges:
small	$[L_{small}, U_{small}]$
medium	$[L_{medium}, U_{medium}]$
large	$[L_{large}, U_{large}]$
force	$[L_{force}, U_{force}]$

Direct effect axioms:

$$(\exists n)[a = \textit{addS}(n) \wedge v = n] \supset \textit{contr}_{small}(a, s) = v$$
$$(\exists n)[a = \textit{releaseS}(n) \wedge v = -n] \supset \textit{contr}_{small}(a, s) = v$$
$$(\exists n)[a = \textit{releaseS}(n) \wedge v = n] \supset \textit{contr}_{medium}(a, s) = v$$
$$(\exists n)[a = \textit{releaseL}(n) \wedge v = -n] \supset \textit{contr}_{large}(a, s) = v$$
$$(\exists n)[a = \textit{exert}(n) \wedge v = n] \supset \textit{contr}_{force}(a, s) = v$$

Preconditions of actions:

$$\textit{Poss}(a, s)$$
$$\textit{conflict}(c, s) = (\exists n, n').[\textit{addS}(n) \in c \wedge \textit{addS}(n') \in c \wedge n \neq n'] \vee$$
$$[\textit{releaseS}(n) \in c \wedge \textit{releaseS}(n') \in c \wedge n \neq n'] \vee$$
$$[\textit{releaseL}(n) \in c \wedge \textit{releaseL}(n') \in c \wedge n \neq n'] \vee$$
$$[\textit{exert}(n) \in c \wedge \textit{exert}(n') \in c \wedge n \neq n']$$

Contribution rules:

$$\textit{medium} \xleftarrow{+} \textit{small} \qquad \textit{large} \xleftarrow{+} \textit{medium}$$
$$\textit{force} \xleftarrow{+-} 2 \times \textit{large} \qquad \textit{force} \xleftarrow{-+} 2 \times \textit{large}$$

Fig. 2. Containers domain described in the extended concurrent situation calculus.

Preconditions of actions We describe preconditions of primitive actions as in [9]. For preconditions of a concurrent action c, we describe by a formula *conflict*(c, s) the conditions under which the primitive actions in c conflict with each other. This is required to handle cases where a set of primitive actions each of which is individually possible may be impossible when executed concurrently.

Ramifications on additive fluents As in the language $\mathcal{C}$+, we consider two kinds of ramifications on numeric-valued fluents, and we express them by acyclic contribution rules (4), where f and h do not contain a situation term.

For instance, Figure 2 shows a formalization of the containers example in this extended version of the concurrent situation calculus. With such a formalization, we can compute the values of fluents, as in Example 1, using GOLOG.

7 Obtaining a Basic Action Theory

From a formalization of an action domain, like in Figure 2, we can obtain a basic action theory in the concurrent situation calculus as follows.

1. We consider the foundational axioms of [9].

2. From the preconditions of primitive actions, conflicts between actions, and range restrictions on additive fluents, we can formalize preconditions of a concurrent action c as in [8], by an axiom of the form

$$\begin{aligned}&\mathit{Poss}(c, s) \equiv\\ &\quad (\exists a)(a \in c) \wedge (\forall a \in c)\mathit{Poss}(a, s) \wedge \neg \mathit{conflict}(c, s) \wedge \mathcal{R}^1[RC(\mathit{do}(c, s))].\end{aligned}$$

Denoted by $\mathcal{R}^1[W]$ is a formula equivalent to the result of applying one step of Reiter's regression procedure [9] on W. We use $RC(s)$ to denote the conjunction of the range constraints on each additive fluent f (i.e., $\bigwedge_f L_f \leq f(s) \leq U_f$) conjoined with additional qualification constraints if given. By this way, a concurrent action is possible if it results in a situation that satisfies the range constraints on additive fluents. For Example 1,

$$\begin{aligned}&RC(s) = L_{\mathit{small}} \leq \mathit{small}(s) \leq U_{\mathit{small}} \wedge L_{\mathit{medium}} \leq \mathit{medium}(s) \leq U_{\mathit{medium}} \wedge\\ &\quad L_{\mathit{large}} \leq \mathit{large}(s) \leq U_{\mathit{large}} \wedge L_{\mathit{force}} \leq \mathit{force}(s) \leq U_{\mathit{force}}.\end{aligned}$$

3. From the direct effect axioms and contribution rules in such a formalization, we can derive successor state axioms for additive fluents by the same kind of transformation in [9], which is based on an explanation closure assumption.
First, we express the cumulative effects of actions on f, by adding the direct and indirect contributions of actions on f, respecting the given range $[L_f, U_f]$. For each additive fluent f, we introduce three new functions: dContr_f, iContr_f, and tContr_f. Intuitively, $\mathit{dContr}_f(\boldsymbol{x}, c, s)$ describes the cumulative direct contributions of primitive actions in c at a situation s:

$$\mathit{dContr}_f(\boldsymbol{x}, c, s) = \sum_{a \in c} \mathit{contr}_f(\boldsymbol{x}, a, s).$$

The indirect contribution of a concurrent action c on f at a situation s is described by $\mathit{iContr}_f(\boldsymbol{x}, c, s)$, relative to a set C of contribution rules:

$$\begin{aligned}&\mathit{iContr}_f(\boldsymbol{x}, c, s) =\\ &\quad \textstyle\sum_{f \xleftarrow{+} \mathcal{E}(h) \in C} \mathcal{E}(\mathit{iContr}_h(\boldsymbol{y}, c, s) + \mathit{dContr}_h(\boldsymbol{y}, c, s) - \mathit{tContr}_h(\boldsymbol{y}, c, s))\\ &- \textstyle\sum_{f \xleftarrow{-} \mathcal{E}(h) \in C} \mathcal{E}(\mathit{iContr}_h(\boldsymbol{y}, c, s) + \mathit{dContr}_h(\boldsymbol{y}, c, s) - \mathit{tContr}_h(\boldsymbol{y}, c, s))\\ &+ \textstyle\sum_{f \xleftarrow{++} \mathcal{E}(h) \in C, \mathit{tContr}_h(\boldsymbol{y}, c, s) > 0} \mathcal{E}(\mathit{tContr}_h(\boldsymbol{y}, c, s))\\ &+ \textstyle\sum_{f \xleftarrow{+-} \mathcal{E}(h) \in C, \mathit{tContr}_h(\boldsymbol{y}, c, s) < 0} \mathcal{E}(\mathit{tContr}_h(\boldsymbol{y}, c, s))\\ &- \textstyle\sum_{f \xleftarrow{-+} \mathcal{E}(h) \in C, \mathit{tContr}_h(\boldsymbol{y}, c, s) > 0} \mathcal{E}(\mathit{tContr}_h(\boldsymbol{y}, c, s))\\ &- \textstyle\sum_{f \xleftarrow{--} \mathcal{E}(h) \in C, \mathit{tContr}_h(\boldsymbol{y}, c, s) < 0} \mathcal{E}(\mathit{tContr}_h(\boldsymbol{y}, c, s)).\end{aligned}$$

For instance, relative to the contribution rules in Figure 2:

$$\mathit{iContr}_{\mathit{medium}}(c, s) = \mathit{iContr}_{\mathit{small}}(c, s) + \mathit{dContr}_{\mathit{small}}(c, s) - \mathit{tContr}_{\mathit{small}}(c, s).$$

After defining direct and indirect contributions of actions on an additive fluent f, we can define the total contribution of actions as follows. If f appears on the right hand side of a contribution rule of form (4), then we add the direct and indirect contributions of actions respecting the range restriction $[L_f, U_f]$:

$$\mathit{tContr}_f(\boldsymbol{x}, c, s) = \begin{cases} U_f - f(\boldsymbol{x}, s) & \text{if } \mathit{sum}_f > U_f - f(\boldsymbol{x}, s) \\ L_f - f(\boldsymbol{x}, s) & \text{if } \mathit{sum}_f < L_f - f(\boldsymbol{x}, s) \\ \mathit{sum}_f & \text{otherwise} \end{cases}$$

where sum_f stands for $\mathit{dContr}_f(\boldsymbol{x}, c, s) + \mathit{iContr}_f(\boldsymbol{x}, c, s)$. Otherwise, the total contribution of actions is simply the sum of the direct and indirect contributions of actions, i.e., sum_f.

Finally, we define the successor state axiom for an additive fluent f:

$$f(\boldsymbol{x}, \mathit{do}(c, s)) = f(\boldsymbol{x}, s) + \mathit{tContr}_f(\boldsymbol{x}, c, s).$$

4. From the given action functions, we can obtain unique names axioms, like $\mathit{addS}(n) \neq \mathit{releaseS}(n')$, etc.
5. We suppose that a description of the initial world is given.

8 Comparing the Two Formalizations

We have described how to formalize an action domain with additive fluents, in two formalisms: the action language $\mathcal{C}$+ and the concurrent situation calculus. We can see in Figures 1 and 2 that two such formalizations look similar. In fact, under some conditions, a formalization D of an action domain in the extended version of $\mathcal{C}$+ and a description I of the initial world can be translated into an action theory $\mathit{sit}(D, I)$ in the extended version of the concurrent situation calculus, such that, for every additive fluent f and for every concurrent action c, the value of f after execution of c is the same according to each formalization.

Suppose that D consists of the following:

– additive fluent constants $F_1, \ldots, F_m$, each F_i with the domain $\{L_{F_i}, ..., U_{F_i}\}$ ($\mathit{Min} \leq L_{F_i}, U_{F_i} \leq \mathit{Max}$); and boolean action constants $A_1, \ldots, A_{m'}$;
– increment laws of form (3) where a is a boolean action constant, f is an additive fluent constant, n is an integer, and ψ is true;
– preconditions of actions of the form

$$\textbf{nonexecutable } a \textbf{ if } \psi \tag{8}$$

where ψ is a conjunction of atoms that does not contain the action constant a.
– acyclic contribution rules of form (4).

Suppose that I consists of the following:

$$0 : F_i = N_i \quad (0 \leq i \leq m)$$

where N_i is an integer in the given range $\{L_{F_i}, \ldots, U_{F_i}\}$, expressing that, at time stamp 0, the value of F_i is N_i.

Then we can obtain $\mathit{sit}(D, I)$ from D and I as follows:

1. For each additive fluent constant $F_i \in D$, declare a corresponding unary additive fluent function $\mathrm{f_i}(s)$ with the range $[L_{\mathrm{f_i}}, U_{\mathrm{f_i}}]$. such that $L_{F_i} = L_{\mathrm{f_i}}$ and $U_{F_i} = U_{\mathrm{f_i}}$. For each boolean action constant $A_i \in D$, declare a corresponding nullary

action function A_i. For instance, for the fluent constant *Small* with the domain $\{L_{Small}, \ldots, U_{Small}\}$ in Figure 1, we declare in Figure 2 the fluent function *small* with the range $[L_{small}, U_{small}]$.
Schemas are frequently used in $\mathcal{C}$+ to represent a large number of constants or statements. For example, *AddS*(n) in the declarations part denotes the action constants *AddS*(*Min*)$, \ldots,$ *AddS*(*Max*). In a situation calculus representation, for such a set of action constants, we can introduce a single action function (e.g., *addS*(n)).

2. For each increment law A_i **increments** F_j **by** N in D, add the formula

$$[a = \mathrm{A}_i \wedge v = N] \supset \mathit{contr}_{\mathrm{f}_j}(a, s) = v. \tag{9}$$

With a function $\mathrm{A}_i(n)$, we can use a single formula to represent all of the formulas (9) for A_i, as seen in Figure 2.

3. Let NEX_F be the set of all causal laws (8) in D such that ψ is a fluent formula. Let $\psi(s)$ be the formula obtained from a fluent formula ψ by replacing every additive fluent atom $F_i = N$ by $\mathrm{f}_i(s) = N$. For each action constant A_i in D, add the formula

$$\mathit{Poss}(\mathrm{A}_i, s) \equiv \bigwedge_{(\textbf{nonexecutable } A_i \textbf{ if } \psi) \in NEX_F} \neg\psi(s).$$

If for every action constant A_i, the right hand side of the equivalence above is $\top$ then we can simply replace all of the equivalences above by the single formula *Poss*(a, s) as in Figure 2 (recall a is implicitly universally quantified.)

4. Let NEX_A be the set of all causal laws (8) in D such that ψ is a formula that contains an action constant. Let $\psi(c, s)$ be the formula obtained from a concurrent action c and a formula ψ by replacing every fluent atom $F_i = N$ with $\mathrm{f}_i(s) = N$, and every action atom A_j (respectively, $\neg A_k$) with $\mathrm{A}_j \in c$ (respectively, $\mathrm{A}_k \notin c$). Then add the following definition:

$$\mathit{conflict}(c, s) \equiv \bigvee_{(\textbf{nonexecutable } A_i \textbf{ if } \psi) \in NEX_A} [\mathrm{A}_i \in c \wedge \psi(c, s)].$$

5. For each contribution rule $F \xleftarrow{\oplus} \mathcal{E}(H)$ in D, add the contribution rule $\mathrm{f} \xleftarrow{\oplus} \mathcal{E}(\mathrm{h})$.
6. For each expression $0 : F_i = N_i$ in I, add the fact $\mathrm{f}_i(S_0) = N_i$.

Suppose that the range $[\mathit{Min}, \mathit{Max}]$ is wide enough that, when compiling D into an action description as described in Section 4, the auxiliary actions *DContr*$_f$, *IContr*$_f$, and *TContr*$_f$ are never undefined due to range violation.

Proposition 1. *Let C be a set of action constants in D and c be the set of corresponding action functions in sit(D, I). Then the following hold:*

(i) C is executable at time stamp 0 with respect to D and I iff Poss(c, S_0) with respect to sit(D, I);

(ii) for every fluent constant F_i, if C is executable at time stamp 0 and $1 : F_i = N_i'$ after the execution of C at time stamp 0, with respect to D and I, then $\mathrm{f}_i(do(c, S_0)) = N_i'$ with respect to sit(D, I).

(iii) for every fluent constant F_i, if Poss(c, S_0) and $\mathrm{f}_i(do(c, S_0)) = N_i'$ with respect to sit(D, I), then $1 : F_i = N_i'$ after the execution of C at time stamp 0, with respect to D and I.

The assumption above is required for the 'if' part of (i), and for (iii).

Although we have incorporated contribution rules into two formalisms in a similar way, and we have shown that, under some conditions, a formalization of an action domain in $\mathcal{C}$+ can be transformed into a formalization in the concurrent situation calculus, these two formalisms are different in general: $\mathcal{C}$+ action descriptions are nonmonotonic and propositional, while the situation calculus action theories are monotonic and first-order. This work can be viewed in part as an attempt to bridge the gap between these two formalisms, in the spirit of [15].

9 Related Work

There are mainly two lines of work related to ours. The first one, [13] and [14], introduces methods to obtain a causal ordering of variables (denoting numeric-valued fluents) from a set of equation-like causal structures, confluence equations and structural equations, each describing a mechanism in a device. Such a causal ordering describes which fluents are directly causally dependent on which other fluents. The goal is, by this way, to understand the causal behavior of a device.

The other line of work, [16] and [8], explicitly represents causal relations among variables by equation-like causal structures, structural equations and contribution equations; so the goal is not to obtain a causal ordering on numeric-valued variables. They use these equations for various problems of reasoning about actions and change. For instance, [16] represents each mechanism with a structural equation, and uses them for modeling counterfactuals. On the other hand, [8] represents each mechanism with a contribution equation, compiles them into an action theory, allowing one to solve problems of reasoning about effects of actions, like planning and prediction.

All [14, 16, 8] suppose that the causal influence among fluents is acyclic. The method of [13] can not in general determine the effects of disturbances by propagation when the causal influences are cyclic. [14, 16] require each variable to be classified as either exogenous or endogenous; the others and we do not.

In our approach, each mechanism is described by a set of contribution rules with the same head. These rules explicitly represent the flow of causal influences among variables; in this sense it can be considered along the second line of work above. Contribution rules are assumed to be acyclic. As in [8], by compiling contribution rules into an action theory, we can solve problems of reasoning about effects of actions. On the other hand, unlike with contribution equations, there is no obvious correspondence between contribution rules and algebraic equations. For instance, in the containers example, with the contribution equations $\mathit{inner}(s) = \mathit{medium}(s) + \mathit{small}(s)$ and $\mathit{total}(s) = \mathit{inner}(s) + \mathit{large}(s)$, one can verify that $\mathit{total}(s) = \mathit{small}(s) + \mathit{medium}(s) + \mathit{large}(s)$. In our approach, we can verify this equation by introducing an auxiliary fluent $\mathit{total}(s)$ and contribution rules for it, but there is no direct correspondence between the equation and the contribution rules. Another difference between contribution equations and contribution rules, is that auxiliary fluents such as $total$ and $inner$ are necessary to write contribution equations, while they are not required in writing contribution rules. This is due to the ability of contribution rules to express more directly the causal influence relationships among fluents. Finally, although contribution equations can handle the first

kind of ramifications mentioned in the introduction, we cannot directly express the second kind of ramifications by them; there is no direct way to describe these ramifications by the other causal structures mentioned above.

10 Conclusion

We have described how to formalize an action domain with additive fluents, in two formalisms: the action language $\mathcal{C}+$ and the concurrent situation calculus. In both cases, first we have extended the formalisms, e.g., by introducing some new constructs or functions and by modifying some axioms. Since some ramifications are not easy to describe in the original formalisms, or using the existing causal structures, we have introduced contribution rules, which express causal influences between additive fluents. After that we have formalized an action domain in the extended versions in four parts: specification of additive fluents with their domains/ranges and actions affecting them, direct effects of actions on additive fluents, preconditions of actions, and ramifications on additive fluents. The formalizations obtained this way can handle not only nonserializable actions, but also ramifications on additive fluents. Investigating the application of our method to other formalisms, such as TAL [17], is a possible future research direction.

Acknowledgments

We thank Selim T. Erdoğan, Joohyung Lee, and Vladimir Lifschitz for helpful comments on an earlier version of the paper. Esra Erdem was supported in part by the Austrian Science Fund (FWF) under project P16536-N04.

Appendix

The Containers Domain presented to CCALC

To be able to solve problems involving ramifications on numeric-valued fluents, we modified the source file `ccalc.pl` of CCALC, so that CCALC can automatically generate domain-independent causal laws presented in Section 4. Then we added some macros to the file `additive` to obtain from increments laws and contribution rules domain-dependent causal laws. The modified source files are available at `http://www.kr.tuwien.ac.at/staff/esra/additive/containers.html`.

We present the containers domain formalization in Figure 1 to CCALC in a file, `containers-domain`, as in Figure 3. We present the problem description in another file, `containers-problem`, as in Figure 4. In these Prolog files, constants start with lower-case letters, and variables with upper-case letters. CCALC computes the values of the fluents after the execution of the concurrent action described in this problem as follows:

```
| ?- query 1.

0:  small=1  medium=1  large=1  force=-2
```

```
ACTIONS:  addS(8)  releaseS(1)  releaseL(2)  exert(8)
    iContr(small)=0  tContr(small)=1  dContr(small)=7
    iContr(medium)=6  tContr(medium)=2  dContr(medium)=1
    iContr(large)=5  tContr(large)=3  dContr(large)=-2
    iContr(force)=-6  tContr(force)=2  dContr(force)=8

1:  small=2  medium=3  large=4  force=0

yes
```

For a more efficient computation, the amount of resources added/released can be declared as "attributes" of the actions, as in Figure 5. To find the solution above, CCALC transforms the formulation of Figure 3 into a propositional theory with 50162 atoms and 216204 clauses (in about 23 minutes), and the computation of the solution above takes 30 seconds. On the other hand, CCALC transforms the formulation of Figure 5 into a propositional theory with 18334 atoms and 90816 clauses (in 26 seconds), and the computation of the solution takes 28 seconds.

The files containing the containers domain description and the problem are also available at `http://www.kr.tuwien.ac.at/staff/esra/additive/containers.html`.

```
% file name: containers-domain

:- include 'additive-cr'.

:- variables
        N,N1 :: nnInteger.

:- constants
        small  :: additiveFluent(l_small..u_small);
        medium :: additiveFluent(l_medium..u_medium);
        large  :: additiveFluent(l_large..u_large);
        force  :: additiveFluent(l_force..u_force);
        addS(nnInteger),
        releaseS(nnInteger),
        releaseL(nnInteger),
        exert(nnInteger) :: exogenousAction.

% direct contributions:

addS(N) increments small by N.
releaseS(N) decrements small by N.
releaseS(N) increments medium by N.
releaseL(N) decrements large by N.
exert(N) increments force by N.

% preconditions:

nonexecutable addS(N) & addS(N1) if N < N1.
nonexecutable releaseS(N) & releaseS(N1) if N < N1.
nonexecutable releaseL(N) & releaseL(N1) if N < N1.
nonexecutable exert(N) & exert(N1) if N < N1.

% contribution rules:

medium <-+- multiply(1,small).
large <-+-  multiply(1,medium).
force <-+-- multiply(2,large).
force <--+- multiply(2,large).
```

Fig. 3. The formalization in Figure 1 presented to CCALC.

```
% file name: containers-problem

:- macros
  maxInt -> 8;  % minInt = -maxInt
  l_small -> 0;   u_small -> 2;
  l_medium -> 0;  u_medium -> 3;
  l_large -> 0;   u_large -> 4;
  l_force -> -8;  u_force -> 8.

:- include 'containers-domain'.

:- query
  label :: 1;
  maxstep :: 1;
  0: small = 1, medium = 1, large = 1, force = -2,
     addS(8), releaseS(1), releaseL(2), exert(8).
```

Fig. 4. The containers problem described in Example 1.

```
% file name: containers-domain-attributes

:- include 'additive-cr'.

:- variables
        N :: nnInteger.

:- constants
        small  :: additiveFluent(l_small..u_small);
        medium :: additiveFluent(l_medium..u_medium);
        large  :: additiveFluent(l_large..u_large);
        force  :: additiveFluent(l_force..u_force);
        addS, releaseS, releaseL, exert :: exogenousAction;
        howmuchToAddS :: attribute(nnInteger) of addS;
        howmuchToReleaseS :: attribute(nnInteger) of releaseS;
        howmuchToReleaseL :: attribute(nnInteger) of releaseL;
        howmuchToExert :: attribute(nnInteger) of exert.

% direct contributions:

addS increments small by N if howmuchToAddS = N.
releaseS decrements small by N if howmuchToReleaseS = N.
releaseS increments medium by N if howmuchToReleaseS = N.
releaseL decrements large by N if howmuchToReleaseL = N.
exert increments force by N if howmuchToExert = N.

% contribution rules:

medium <-+- multiply(1,small).
large <-+-  multiply(1,medium).
force <-+-- multiply(2,large).
force <--+- multiply(2,large).
```

Fig. 5. The containers domain presented to CCALC using attributes.

The Containers Domain presented to GOLOG

Additional definitions and axioms that need to be added to a Golog interpreter, to handle additive fluents and contribution rules, are contained in the file `additive-cr.pl` available at `http://www.kr.tuwien.ac.at/staff/esra/additive/containers.html`.

Figure 6 shows the implementation of the Containers domain formalization in the concurrent situation calculus. The set of initial values (initial state) and the range restrictions on the four fluents as used in Example 1 is shown in Figure 7. The following is a sample query for this problem and the answers:

```
[eclipse 3]: C=[addS(8),releaseS(1),releaseL(2),exert(8)],
             value(F,V,do(C,s0)).

C = [addS(8), releaseS(1), releaseL(2), exert(8)]
F = small
V = 2
Yes (0.00s cpu, solution 1, maybe more) ? ;

C = [addS(8), releaseS(1), releaseL(2), exert(8)]
F = medium
V = 3
Yes (0.00s cpu, solution 2, maybe more) ? ;

C = [addS(8), releaseS(1), releaseL(2), exert(8)]
F = large
V = 4
Yes (0.00s cpu, solution 3, maybe more) ? ;

C = [addS(8), releaseS(1), releaseL(2), exert(8)]
F = force
V = 0
Yes (0.00s cpu, solution 4, maybe more) ? ;

No (0.00s cpu)
```

```
primitive_action(addS(N)).         primitive_action(releaseS(N)).
primitive_action(releaseL(N)).     primitive_action(exert(N)).

contr(small, A, V, S):- A=addS(N), V is N;
                        A=releaseS(N), V is -N;
                        not (A=addS(N) ; A=releaseS(N)), V is 0.

contr(medium,A,V,S):- A=releaseS(N), V is N;
                      not A=releaseS(N), V is 0.

contr(large,A,V,S):- A=releaseL(N), V is -N;
                     not A=releaseL(N), V is 0.

contr(force,A,V,S):- A=exert(N), V is N;
                     not A=exert(N), V is 0.

value(small,V,S):- small(V,S).
value(medium,V,S):- medium(V,S).
value(large,V,S):- large(V,S).
value(force,V,S):- force(V,S).

%% Succ State Axioms

small(V,do(C,S)):- small(Vs,S), tContr(small,C,Vc,S), V is Vs + Vc.

medium(V,do(C,S)):- medium(Vs,S), tContr(medium,C,Vc,S), V is Vs + Vc.

large(V,do(C,S)):- large(Vs,S), tContr(large,C,Vc,S), V is Vs + Vc.

force(V,do(C,S)):- force(Vs,S), tContr(force,C,Vc,S), V is Vs + Vc.

%% All primitive actions are always possible

poss(A,S):- primitive_action(A).

conflict(C,S):- member(addS(N),C), member(addS(N1),C), not N=N1;
                member(releaseS(N),C), member(releaseS(N1),C), not N=N1;
                member(releaseL(N),C), member(releaseL(N1),C), not N=N1;
                member(exert(N),C), member(exert(N1),C), not N=N1.

medium <-+- 1 * small.
large <-+- 1 * medium.
force <-+-- 2 * large.
force <--+- 2 * large.
foo <--- bar.
```

Fig. 6. Implementation of the formalization in Figure 2.

```
lower(small,0). lower(medium,0). lower(large,0). lower(force,-8).
upper(small,2). upper(medium,3). upper(large,4). upper(force,8).

small(1,s0). medium(1,s0). large(1,s0). force(-2,s0).
```

Fig. 7. The containers problem described in Example 1.

Proof of Proposition 1

Let $\mathbf{F}$ be the set of all additive fluent constants $F_1, \ldots, F_m$. Let $\mathbf{A}$ be the set of all boolean action constants $A_1, \ldots, A_{m'}$.

Definition 1. *A set C of boolean action constants in $\mathbf{A}$ is* executable *at a state S defined by an interpretation $F_i = N_i$ of fluent constants in $\mathbf{F}$ such that $L_{F_i} \leq N_i \leq U_{F_i}$, if*

(i) for every fluent constant F_i in $\mathbf{F}$, $L_{F_i} \leq N_i + TContr(F_i) \leq U_{F_i}$ and
*(ii) for every causal law (**nonexecutable** A_i **if** ψ) in D, $S \cup C$ does not satisfy $A_i \wedge \psi$.*

Suppose that the range $[Min, Max]$ is wide enough that, when compiling D into an action description as described in Section 3, the auxiliary actions $DContr_f$, $IContr_f$, and $TContr_f$ are never undefined due to range violation.

We use the following lemmas and notation to prove the proposition above. Let S_I denote the initial state defined by atoms

$$F_i = N_i \quad (F_i \in \mathbf{F}, L_{F_i} \leq N_i \leq U_{F_i})$$

where every $0 : F_i = N_i$ is in I. Remember that the initial situation S_0 is defined by the set of facts $\mathrm{f_i}(S_0) = N_i$ where $0 : F_i = N_i$ is in I.

Lemma 1. *Condition (ii) of Definition 1 holds for the initial state S_I iff*

(i) for every action A_i in c, $sit(D, I) \models Poss(\mathrm{A}_i, S_0)$; and
(ii) $sit(D, I) \models \neg conflict(c, S_0)$.

Proof: Take any causal law (**nonexecutable** A_i **if** ψ) in D. If ψ is a fluent formula, $S_I \cup C \not\models A_i \wedge \psi$ iff $S_I \not\models \psi$ iff $\neg\psi(S_0)$ iff $Poss(\mathrm{A}_i, S_0)$; moreover, since ψ does not contain an action constant, $\neg conflict(c, S_0)$. Otherwise, i.e., ψ contains an action constant, the following holds: $S_I \cup C \not\models A_i \wedge \psi$ iff $A_i \notin C$ or $S_I \cup C \not\models \psi$ iff $\neg\psi(c, S_0)$ iff $\neg conflict(c, S_0)$; moreover, since ψ contains an action constant, $Poss(\mathrm{A}_i, S_0)$.

Lemma 2. *If C is executable at a state S, then there exists a unique transition $\langle S, Z, S' \rangle$ in the transition diagram described by D, such that the following hold for every additive fluent constant F_j in $\mathbf{F}$:*

(i) for every boolean action constant $A_i \in C$, for some integer Y, $Contr(A_i, F_j) = Y$ is in Z;
(ii) for some integer Y, $DContr(F_j) = Y$ is in Z;
(iii) for some integer Y, $IContr(F_j) = Y$ is in Z;
(iv) for some integer Y, $TContr(F_j) = Y$ is in Z.

Proof: Follows from the definition of the transition diagram described by the action description obtained from D. First of all, if C is executable at S then there is a transition $\langle S, Z, S' \rangle$ in the transition diagram described by D. Since the effects of all actions are deterministic (they either increment or decrement the values of additive fluents by a certain amount), and the causal influence among fluents is acyclic via contribution rules, such a transition is unique. (i) Due to the translation of increments laws into causal laws, for every A_i and F_j, if they appear in an increments law

A_i **increments** F_j **by** N

in D, *Contr*$(A_i, F_j) = N$ is in Z; otherwise, *Contr*$(A_i, F_j) = 0$ is in Z. (ii)–(iv) If C is executable at S then, due to Definition 1(i), for every fluent constant F_j, there exist some integer N_j, such that *TContr*$(F_j) = N_j$ is in Z. Then, due to the definition of *TContr* obtained by the translation of contribution rules into causal laws, there exist some integers Y and Y' such that *DContr*$(F_j) = Y$ and *IContr*$(F_j) = Y'$ are in Z.

Let C_R denote the concurrent action X in the unique transition described in the lemma above.

Lemma 3. *If C is executable at S_I, then, for every fluent constant F_j in* **F***, for every corresponding fluent function name* $\mathrm{f_j}$ *in sit(D, I), and for every integer X in $\{Min, \ldots, Max\}$, the following hold:*

(i) For every boolean action constant A_i in C, and for every corresponding action function $\mathrm{A_i}$ *in sit(D, I),*
Contr$(A_i, F_j) = X \in C_R$ iff contr$_{\mathrm{f_j}}(\mathrm{A_i}, S_0) = X$;
(ii) DContr$(F_j) = X \in C_R$ iff dContr$_{\mathrm{f_j}}(c, S_0) = X$;
(iii) IContr$(F_j) = X \in C_R$ iff iContr$_{\mathrm{f_j}}(c, S_0) = X$;
(iv) TContr$(F_j) = X \in C_R$ iff tContr$_{\mathrm{f_j}}(c, S_0) = X$.

Proof: (i) By Lemma 2(i), there exists a transition $\langle S_I, Z, S' \rangle$ in the transition diagram described by the action description obtained from D such that, for every boolean action constant $A_i \in C$, for some integer X, *Contr*$(A_i, F_j) = X$ is in Z. On the other hand, due to the formulas added at Step 2 of the transformation and the definitional axiom of *contr*$_{\mathrm{f_j}}$, for every corresponding $\mathrm{A_i}$, *contr*$_{\mathrm{f_j}}(\mathrm{A_i}, S_0) = X$. (ii)–(iv) Follow from (i), the definitions of *DContr*$(F_j) = X$, *IContr*$(F_j) = X$, *TContr*$(F_j) = X$ in the action description obtained from D (under the assumption that the range $[Min, Max]$ is wide enough that, when compiling D into an action description, the auxiliary actions *DContr*$_f$, *IContr*$_f$, and *TContr*$_f$ are never undefined due to range violation), and the definitions of *dContr*$_{\mathrm{f_j}}$, *iContr*$_{\mathrm{f_j}}$, *tContr*$_{\mathrm{f_j}}$ in the basic action theory obtained from *sit*(D, I).

Lemma 4. *If C is executable at S_I, then, for every fluent constant F_i in* **F***, the following hold:*

for some integer X, TContr$(F_i) = X \in C_R$ and $L_{F_i} \leq X + N_i \leq U_{F_i}$ according to D and I
iff
$\mathcal{R}^1[RC(do(c, S_0))]$ *according to sit(D, I).*

Proof: Follows from Lemma 3, the definition of RC, and the correctness theorem of the regression operator $\mathcal{R}$ [9].

Lemma 5. *If $sit(D, I) \models Poss(c, S_0)$, then condition (i) of Definition 1 for the initial state S_I holds.*

Proof: $Poss(c, S_0) \supset R^1[RC(do(c, S_0))] \supset RC(do(c, S_0)) \supset L_{\mathrm{f_i}} \leq \mathrm{f_i}(do(c, S_0)) \leq U_{\mathrm{f_i}} \supset L_{\mathrm{f_i}} \leq \mathrm{f_i}(S_0) +$ *tContr*$_{\mathrm{f_i}}(c, S_0) \leq U_{\mathrm{f_i}}$. Then, by Lemma 3, $L_{F_i} \leq N_i +$ *TContr*$(F_i) \leq U_{F_i}$.

Lemma 6. *C is executable at S_I iff $sit(D, I) \models Poss(c, S_0)$.*

Proof: ($\Rightarrow$) Follows from Lemmas 1 and 4. ($\Leftarrow$) By Lemma 5, condition (i) holds. $Poss(c, S_0)$ implies $Poss(\mathrm{A}_i, S_0)$ and $\neg conflict(c, S_0)$. Then by Lemma 1 condition (ii) holds.

Lemma 7. *If C is executable at S_I, then, for every fluent constant F_j in* **F**, *for every corresponding fluent function name $\mathrm{f_j}$ in sit(D, I), and for every integer N_i' in $\{$Min, ..., Max$\}$, the following hold: $1 : F_j = N_i'$ after the execution of C at S_I iff $sit(D, I) \models \mathrm{f}_i(do(c, S_0)) = N_i'$.*

Proof: ($\Rightarrow$) By Lemma 2, if C is executable at S_I then $\langle S_I, C_R, S' \rangle$ in the transition diagram described by D, such that, for some integer X, *TContr*$(F_i) = X$ is in C_R. Then, by the definition of the transition diagram described by D, $F_i = N_i + X$ is in S' where $F_i = N_i$ is in S_I. On the other hand, by Lemma 3, $sit(D, I) \models$ *tContr*$_{\mathrm{f}_i}(c, S_0) = X$. Then, by the successor state axiom of f_i in the basic action theory obtained from $sit(D, I)$, $\mathrm{f}_i(do(c, S_0)) = \mathrm{f}_i(S_0) +$ *tContr*$_{\mathrm{f}_i}(c, S_0) = N_i + X = N_i'$. ($\Leftarrow$) By the successor state axiom of f_i in the basic action theory obtained from $sit(D, I)$, $sit(D, I) \models N_i' = \mathrm{f}_i(S_0) +$ *tContr*$_{\mathrm{f}_i}(c, S_0)$. By Lemma 3, for some integer X, *tContr*$_{\mathrm{f}_i}(c, S_0) = X$ iff *TContr*$(F_i) = X \in C_R$. And by the definition of $sit(D, I)$, $\mathrm{f}_i(S_0) = N_i$ iff $0 : F_i = N_i$ is in I. On the other hand, by the definition of the transition diagram described by D, $1 : F_i = N_i + X$ holds.

Proof of Proposition 1: (i)–(iii) hold by Lemmas 6 and 7.

References

1. Koehler, J.: Planning under resource constraints. In: Proc. ECAI. (1998) 489–493
2. Kvarnström, J., Doherty, P., Haslum, P.: Extending TALplanner with concurrency and resources. In: Proc. ECAI. (2000) 501–505
3. Bacchus, F., Ady, M.: Planning with resources and concurrency: A forward chaining approach. In: Proc. IJCAI. (2001) 417–424
4. Lee, J., Lifschitz, V.: Describing additive fluents in action language $\mathcal{C}+$. In: Proc. IJCAI. (2003)
5. Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., Turner, H.: Nonmonotonic causal theories. AIJ **153** (2004) 49–104
6. Baral, C., Son, T.C., Tuan, L.: A transition function based characterization of actions with delayed and continuous effects. In: Proc. KR. (2002)
7. Gelfond, M., Lifschitz, V.: Action languages. ETAI **3** (1998) 195–210
8. Erdem, E., Gabaldon, A.: Cumulative effects of concurrent actions on numeric-valued fluents. In: Proc. AAAI. (2005) 627–632
9. Reiter, R.: Knowledge in action: Logical Foundations for specifying and implementing dynamical systems. MIT Press (2001)
10. Lin, F.: Embracing causality in specifying the indirect effects of actions. In: Proc. IJCAI. (1995) 1985–1991
11. McCain, N., Turner, H.: A causal theory of ramifications and qualifications. In: Proc. IJCAI. (1995) 1978–1984
12. Thielscher, M.: Ramification and causality. AIJ **89** (1997) 317–364
13. de Kleer, J., Brown, J.S.: A qualitative physics based on confluences. AIJ **24** (1984) 7–83
14. Iwasaki, Y., Simon, H.: Causality in device behavior. AIJ **29** (1986) 3–32
15. Giunchiglia, E., Lifschitz, V.: Action languages, temporal action logics and the situation calculus. In: Proc. NRAC. (1999)
16. Halpern, J., Pearl, J.: Causes and explanations: a structural-model approach–Part I: Causes. In: Proc. UAI. (2001) 194–202
17. Doherty, P., Gustafsson, J., Karlsson, L., Kvarnstrom, J.: (TAL) Temporal Action Logics: Language specification and tutorial. ETAI **2** (1998) 273–306