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

Joohyung Lee and Vladimir Lifschitz
Department of Computer Sciences,
University of Texas, Austin, TX 78712
{appsmurf,vl}@cs.utexas.edu

Abstract

An additive fluent is a fluent with numerical values such that the effect of several concurrently executed actions on it can be computed by adding the effects of the individual actions. We propose a method for describing effects of actions on additive fluents in the declarative language $\mathcal{C}$+. An implementation of this language, called the Causal Calculator, can be used for the automation of examples of commonsense reasoning involving additive fluents.

1 Introduction

Action languages [Gelfond and Lifschitz, 1998] serve to describe effects of actions on the states of the world. For instance, the expression

$$\mathit{Walk}(x,l) \ \textbf{causes}\ \mathit{Location}(x)\!=\!l \qquad (1)$$

is a proposition, or "causal law," of action language $\mathcal{C}$+ [Giunchiglia *et al.*, 2003] that describes an effect of action $\mathit{Walk}(x,l)$: this action causes the location of x to become l. The semantics of action languages is defined in terms of "transition systems"—directed graphs whose vertices correspond to the states of the world, and whose edges correspond to the execution of actions. The execution of a sequence of actions can be represented by a path in such a graph.

Some action languages, including $\mathcal{C}$+, allow us to talk about the effects of the concurrent execution of actions. Causal law (1) is understood in $\mathcal{C}$+ to imply that $\mathit{Location}(x) = l$ holds after any event that involves the execution of $\mathit{Walk}(x,l)$, even if other actions are executed concurrently. To distinguish the events involving the concurrent execution of actions a_1 and a_2 from the events that involve a_1 but not a_2, we can write

$$a_1 \wedge a_2 \ \textbf{causes}\ \ldots,$$
$$a_1 \wedge \neg a_2 \ \textbf{causes}\ \ldots.$$

In this paper we investigate the possibility of using $\mathcal{C}$+ to represent the effects of actions on fluents of a special kind, called "additive." An additive fluent is a fluent with numerical values such that the effect of several concurrently executed actions on it can be computed by adding the effects of the individual actions. For example, the gross receipts of a store are represented by an additive fluent: when several customers pay to different cashiers simultaneously, the gross receipts will increase by the sum of the "contributions" of the individual customers. The voltage of a battery is an additive fluent: the increase in voltage obtained by adding several cells to a battery can be computed by addition. In mechanics, the velocity of a particle is an additive fluent, because the net effect of several forces on this fluent over a time interval equals the sum of the effects of the individual forces. Additive fluents are ubiquitous; this may be the reason why adding numbers is such a useful operation.

Unfortunately, the **causes** construct of $\mathcal{C}$+ and similar languages is not directly applicable to describing the effects of actions on additive fluents. Consider, for instance, the effect of the action $\mathit{Buy}(x,n)$ (customer x buys n books) on the number of books available at the bookstore. The causal law

$$\mathit{Buy}(x,n)\ \textbf{causes}\ \mathit{InStore}\!=\!k\!-\!n\ \textbf{if}\ \mathit{InStore}\!=\!k \qquad (2)$$

is applicable in the case when no customer other than x is buying books at the same time: $k-n$ books are available after the event if there were k books in the store before the event. But (2) is not acceptable if we are interested in the concurrent execution of such actions.

We introduce here a syntactic construct, **increments**, that allows us to describe the effects of actions on additive fluents. Semantically this construct is treated as "syntactic sugar" on top of $\mathcal{C}$+: the propositions involving that construct are viewed as abbreviations for causal laws of $\mathcal{C}$+. The interpretation of **increments** described below has been used to extend the implementation of $\mathcal{C}$+, called the Causal Calculator (CCALC), to cover additive fluents.

After a review of transition systems and of the syntax of $\mathcal{C}$+ in the next two sections, we describe the syntax of increment laws (Section 4), define their semantics by showing how to treat them as abbreviations (Section 5), and illustrate the use of additive fluents in the language of CCALC by formalizing an example that involves buying and selling (Section 6). Two other examples of commonsense reasoning related to additive fluents are discussed in Sections 7 and 8. A proposition stated in Section 9 confirms that additive fluents behave as one would expect on the basis of the informal discussion of additivity above, and thus provide a justification for the approach to formalizing additive fluents proposed in this paper.

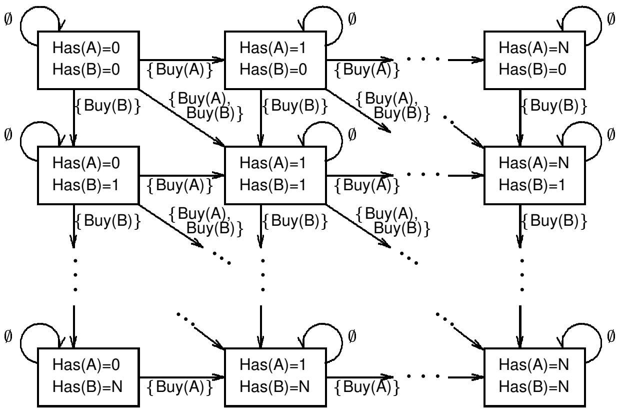

Figure 1: A transition system.

2 Transition Systems

Consider a set of symbols, called *constants*, along with a nonempty finite set of symbols $Dom(c)$ assigned to each constant c. We call $Dom(c)$ the *domain* of c.

The constants are divided into two groups—*fluent constants* and *action constants*. Intuitively, a fluent constant represents a fluent, and the elements of its domain are the possible values of that fluent. A state of the world is characterized by a function that maps each fluent constant to an element of its domain. In [Giunchiglia *et al.*, 2003, Section 4.2], fluent constants are further subdivided into *simple* and *statically determined*; in the examples below, only fluent constants of the first kind will be used.

A function that maps each action constant to an element of its domain characterizes an event occurring over a time interval between two successive states. In many examples of the use of $\mathcal{C}$+ the domain of every action constant is the set of truth values $\{\mathsf{f}, \mathsf{t}\}$; constants with this domain are called *Boolean*. Intuitively, a Boolean action constant represents an action; the value of such a constant is t if the action is one of those that are executed as part of the event.

A *transition system* is a directed graph whose vertices are functions that map every fluent constant to an element of its domain, and whose edges are labeled by functions that map every action constant to an element of its domain. The vertices of a transition system are called its *states*. The functions labeling the edges of a transition system are called *events*. As an example, consider a transition system representing the effect of buying a book on the number of books that the person owns (Figure 1). It uses two fluent constants—$Has(A)$ (the number of books that Alice has) and $Has(B)$ (the number of books that Bob has)—with the domain $\{0, \ldots, N\}$, where N is a fixed nonnegative integer, and two Boolean action constants—$Buy(A)$ (Alice buys a book) and $Buy(B)$ (Bob buys a book). Every state is represented by two equations showing the values of the fluent constants. Every event is represented by the set of action constants that are mapped to t. The loops are labeled by the trivial event $\emptyset$ (no actions are executed). The horizontal edges are labeled by the event in which Alice buys a book and Bob doesn't; along each of the vertical edges, Bob buys a book and Alice doesn't. The diagonal edges correspond to Alice and Bob buying books concurrently.

3 Syntax of $\mathcal{C}$+

The review of the syntax of $\mathcal{C}$+ in this section follows [Giunchiglia *et al.*, 2003, Section 4.2]. An *atom* is an expression of the form $c = v$, where c is a constant and $v \in Dom(c)$. For instance, the equalities in Figure 1 are atoms. A *formula* is a propositional combination of atoms. By a *fluent formula* we mean a formula such that all constants occurring in it are fluent constants. An *action formula* is a formula that contains at least one action constant and no fluent constants.

A *static law* is an expression of the form

$$\mathbf{caused}\ F\ \mathbf{if}\ G \tag{3}$$

where F and G are fluent formulas. An *action dynamic law* is an expression of the form (3) in which F is an action formula and G is a formula. A *fluent dynamic law* is an expression of the form

$$\mathbf{caused}\ F\ \mathbf{if}\ G\ \mathbf{after}\ H \tag{4}$$

where F and G are fluent formulas and H is a formula, provided that F does not contain statically determined constants. A *causal law* is a static law, or an action dynamic law, or a fluent dynamic law.

Here are two examples. The expression

$$Buy(x)\ \mathbf{causes}\ Has(x) = k+1\ \mathbf{if}\ Has(x) = k \tag{5}$$

where $x \in \{A, B\}$ and $k \in \{0, ..., N-1\}$, is an abbreviation for the fluent dynamic law

$$\mathbf{caused}\ Has(x) = k+1\ \mathbf{if}\ \top\ \mathbf{after}\ Buy(x) = \mathsf{t} \wedge Has(x) = k$$

($\top$ is the 0-place connective "true"). The expression

$$\mathbf{nonexecutable}\ Buy(x)\ \mathbf{if}\ Has(x) = N \tag{6}$$

is an abbreviation for the fluent dynamic law

$$\mathbf{caused}\ \bot\ \mathbf{if}\ \top\ \mathbf{after}\ Buy(x) = \mathsf{t} \wedge Has(x) = N$$

($\bot$ is the 0-place connective "false").

Causal laws (5) and (6) describe an effect of action $Buy(x)$ and a restriction on its executability. To get a complete description of Figure 1 in language $\mathcal{C}$+, we need two more postulates. First, we should specify that the edge labels of the transition system may assign truth values to the action constants $Buy(x)$ arbitrarily. This is expressed by

$$\mathbf{exogenous}\ Buy(x), \tag{7}$$

which stands for the pair of action dynamic laws

$$\mathbf{caused}\ Buy(x) = \mathsf{f}\ \mathbf{if}\ Buy(x) = \mathsf{f},$$
$$\mathbf{caused}\ Buy(x) = \mathsf{t}\ \mathbf{if}\ Buy(x) = \mathsf{t}.$$

Action $Buy(x)$ is exogenous in the sense that the action description does not determine whether that action is executed. If it is not executed then there is a cause for this; if it is executed then there is a cause for that too. Second, we need to say that the fluent constant $Has(x)$ satisfies the "commonsense law of inertia": in the absence of evidence to the contrary, its value after an event is assumed to be the same as its value before the event. This is expressed by

$$\mathbf{inertial}\ Has(x), \tag{8}$$

which stands for $N+1$ fluent dynamic laws

$$\textbf{caused } \mathit{Has}(x)=k \textbf{ if } \mathit{Has}(x)=k \textbf{ after } \mathit{Has}(x)=k$$

($k \in \{0,...,N\}$). If the value of $\mathit{Has}(x)$ after an event is the same as the value before the event then there is a cause for this. Intuitively, inertia is the cause.

An *action description* is a set of causal laws. The semantics of $\mathcal{C}$+, described in [Giunchiglia *et al.*, 2003, Section 4.4], specifies the transition system represented by any given action description. For instance, action description (5)–(8) represents the transition system shown in Figure 1.

This action description does not say explicitly that the trivial event $\emptyset$ has no effect on the values of $\mathit{Has}(A)$ and $\mathit{Has}(B)$, or that event $\{\mathit{Buy}(A)\}$ does not affect the value of $\mathit{Has}(B)$. Nevertheless, every edge of the transition system labeled $\emptyset$ is a loop, and every edge labeled $\{\mathit{Buy}(A)\}$ is horizontal, because of the postulates (8) that express, under the semantics of $\mathcal{C}$+, the persistence property of $\mathit{Has}(x)$.

Similarly, causal laws (5)–(8) do not say anything about the concurrent execution of actions $\mathit{Buy}(A)$ and $\mathit{Buy}(B)$. But the edges labeled $\{\mathit{Buy}(A), \mathit{Buy}(B)\}$ in Figure 1 are directed diagonally, in accordance with our commonsense expectations. This fact illustrates the convenience of the approach to concurrency incorporated in the semantics of $\mathcal{C}$+.

However, as discussed in the introduction, this built-in mechanism is not directly applicable to the effects of actions on additive fluents. We are now ready to turn to the main subject of this paper—extending $\mathcal{C}$+ with the additional notation that resolves this difficulty.

4 Increment Laws

In our proposed extension of $\mathcal{C}$+, some of the simple fluent constants can be designated as *additive*. The domain of every additive fluent constant is assumed to be a finite set of numbers. We understand "numbers" as (symbols for) elements of any set with an associative and commutative binary operation $+$ that has a neutral element 0.[1] Effects of actions on additive fluents are described in extended $\mathcal{C}$+ by causal laws of a new kind—"increment laws." Accordingly, we modify the definition of a causal law reproduced in Section 3 in two ways. First, in causal laws (3) and (4) formula F is not allowed to contain additive fluent constants. Second, we extend the class of causal laws by including *increment laws*—expressions of the form

$$a \textbf{ increments } c \textbf{ by } n \textbf{ if } G \tag{9}$$

where

- a is a Boolean action constant,
- c is an additive fluent constant,
- n is a number, and
- G is a formula that contains no Boolean action constants.

[1]The additive group of integers is the main example we are interested in, and this is the case that has been implemented. The $\max$ operation on an ordered set with the smallest element is another interesting case.

Notation: x ranges over $\{A, B\}$.

Action constants:	Domains:
$\mathit{Buy}(x)$	Boolean
Additive fluent constant:	Domain:
$\mathit{InStore}$	$\{0,\ldots,N\}$

Causal laws:
$\mathit{Buy}(x)$ **increments** $\mathit{InStore}$ **by** -1
exogenous $\mathit{Buy}(x)$

Figure 2: An action description in extended $\mathcal{C}$+.

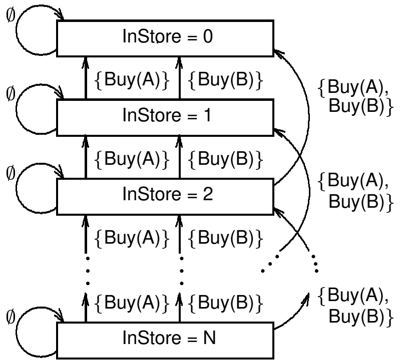

Figure 3: The transition system described by Figure 2.

We will drop '**if** G' in (9) if G is $\top$. In extended $\mathcal{C}$+, an *action description* is a set of causal laws that contains finitely many increment laws.

In the next section we define the semantics of extended $\mathcal{C}$+ by describing a translation that eliminates increment laws in favor of additional action constants.

As an example, consider the effects of actions $\mathit{Buy}(A)$, $\mathit{Buy}(B)$ on the number of books available in the bookstore where Alice and Bob are buying books. A description of these effects in extended $\mathcal{C}$+ is shown in Figure 2 (as before, N is a fixed nonnegative integer). The transition system represented by the translation of Figure 2 into the non-extended language $\mathcal{C}$+ is depicted in Figure 3 (with the auxiliary action constants dropped from the edge labels). The causal laws in Figure 2 do not say explicitly that the trivial event $\emptyset$ has no effect on the value of *InStore*, or that the concurrent execution of actions $\mathit{Buy}(A)$ and $\mathit{Buy}(B)$ decrements the value of this fluent by 2. Nevertheless, every edge of the corresponding transition system labeled $\emptyset$ is a loop, and every edge labeled $\{\mathit{Buy}(A), \mathit{Buy}(B)\}$ goes up 2 levels, in accordance with our commonsense expectations. This happens because Figure 2 classifies *InStore* as an additive fluent constant.

The causal laws in this action description do not say explicitly that actions $\mathit{Buy}(x)$ are not executable when $\mathit{InStore}=0$, and that actions $\mathit{Buy}(A)$, $\mathit{Buy}(B)$ cannot be executed concurrently when $\mathit{InStore}=1$. This is taken care of by our semantics of increment laws, in view of the fact that the domain of *InStore* does not contain negative numbers.

5 Translating Increment Laws

Let D be an action description in extended $\mathcal{C}$+. In connection with the increment laws (9) in D, the following terminology will be used: about the Boolean action constant a, the additive fluent constant c and the number n in (9) we will say that a is a *c-contributing* constant, and that n is a *contribution* of a to c.

The auxiliary action constants introduced in the translation are expressions of the form $\mathit{Contr}(a, c)$, where c is an additive fluent constant, and a is a c-contributing action constant. The domain of $\mathit{Contr}(a, c)$ consists of all contributions of a to c and number 0.

To translate the increment laws from D, we

(i) replace each increment law (9) in D with the action dynamic law

$$\textbf{caused } \mathit{Contr}(a, c) = n \textbf{ if } a = \mathrm{t} \wedge G, \tag{10}$$

(ii) for every auxiliary constant $\mathit{Contr}(a, c)$, add the action dynamic law

$$\textbf{caused } \mathit{Contr}(a, c) = 0 \textbf{ if } \mathit{Contr}(a, c) = 0, \tag{11}$$

(iii) add the fluent dynamic laws

$$\begin{array}{r} \textbf{caused } c = v + \sum_a v_a \textbf{ if } \top \\ \textbf{after } c = v \wedge \bigwedge_a \mathit{Contr}(a, c) = v_a \end{array} \tag{12}$$

for every additive fluent constant c, every $v \in \mathit{Dom}(c)$, and every function $a \mapsto v_a$ that maps each c-contributing constant a to an element of the domain of $\mathit{Contr}(a, c)$ so that $v + \sum_a v_a$ is in the domain of c.

The sum and the multiple conjunction in (12) range over all c-contributing constants a.

Causal law (10) interprets increment law (9) as the assertion that executing a (possibly along with other actions) causes constant $\mathit{Contr}(a, c)$ to get the value n, under some conditions characterized by formula G. Causal laws (11) say that the value of this constant is 0 by default, that is to say, when another value is not required by any increment law. Causal laws (12) say that the value of an additive fluent constant after an event can be computed as the sum of the value of this constant prior to the event and the contributions of all actions to this constant.

The result of translating increment laws from Figure 2 is shown in Figure 4. In this case, the translation described above introduces two auxiliary action constants: $\mathit{Contr}(\mathit{Buy}(A), \mathit{InStore})$ and $\mathit{Contr}(\mathit{Buy}(B), \mathit{InStore})$. The domain of each of them has 2 elements: the contribution -1 of $\mathit{Buy}(x)$ to $\mathit{InStore}$ and number 0.

The edges of the transition system described by Figure 4, and the corresponding events, can be computed using the methods presented in [Giunchiglia *et al.*, 2003, Sections 4.2, 2.6]. (This calculation involves turning the action description into a definite causal theory and computing the models of the completion of this theory.) Every event assigns values to each action constant, including the auxiliary constants $\mathit{Contr}(\mathit{Buy}(x), \mathit{InStore})$. For instance, the label

$$\{\mathit{Buy}(A), \mathit{Buy}(B)\}$$

Notation: x ranges over $\{A, B\}$.

Action constants:	Domains:
$\mathit{Buy}(x)$	Boolean
$\mathit{Contr}(\mathit{Buy}(x), \mathit{InStore})$	$\{-1, 0\}$

Additive fluent constant:	Domain:
$\mathit{InStore}$	$\{0, \dots, N\}$

Causal laws:

caused $\mathit{Contr}(\mathit{Buy}(x), \mathit{InStore}) = -1$ **if** $\mathit{Buy}(x) = \mathrm{t}$
caused $\mathit{Contr}(\mathit{Buy}(x), \mathit{InStore}) = 0$ **if** $\mathit{Contr}(\mathit{Buy}(x), \mathit{InStore}) = 0$
caused $\mathit{InStore} = v + v_1 + v_2$ **if** $\top$ **after** $\mathit{InStore} = v \wedge$
$\mathit{Contr}(\mathit{Buy}(A), \mathit{InStore}) = v_1 \wedge \mathit{Contr}(\mathit{Buy}(B), \mathit{InStore}) = v_2$
for all $v \in \{0, \dots, N\}$ and $v_1, v_2 \in \{-1, 0\}$
such that $v + v_1 + v_2 \geq 0$
exogenous $\mathit{Buy}(x)$

Figure 4: The result of translating increment laws from Figure 2.

in Figure 3 represent the following event E:

$$E(\mathit{Buy}(A)) = \mathrm{t},\ E(\mathit{Contr}(\mathit{Buy}(A), \mathit{InStore})) = -1,$$
$$E(\mathit{Buy}(B)) = \mathrm{t},\ E(\mathit{Contr}(\mathit{Buy}(B), \mathit{InStore})) = -1.$$

The Causal Calculator, which implements $\mathcal{C}$+ with **increment** laws, can be downloaded from

`http://www.cs.utexas.edu/users/tag/ccalc/` .

6 Reasoning about Money

As an application of these ideas to automated commonsense reasoning, consider the following example:

> I have \$6 in my pocket. A newspaper costs \$1, and a magazine costs \$3. Can I buy 2 newspapers and a magazine? A newspaper and 2 magazines?

These questions are about the executability of some concurrently executed actions, and the answers are determined by the effect of these actions on an additive fluent—the amount of money that I have.

Figure 5 describes the relevant properties of buying and selling in the input language of CCALC. There are objects of four sorts in this domain: agents, resources, items (to be purchased) and (nonnegative) integers; items are a subset of resources. The buyer and the seller are agents; money is a resource; $0, \dots,$ `maxInt` are integers. The price of an item is an integer. The number of units of a resource that an agent has is an integer-valued additive fluent. Buying is an exogenous action. The four causal laws that follow these declarations are self-explanatory; `decrements` is an abbreviation defined in terms of `increments`. The last causal law expresses that the number of units that are being purchased is uniquely defined.

Figure 6 expresses the first of the two questions stated at the beginning of this section. The line `maxstep :: 1` tells CCALC that the query is about paths of length 1 in the transition system. The question is whether the transition system contains an edge that begins in a state in which the buyer

```
:- sorts
  agent;  resource >> item;  nnInteger.

:- variables
  Ag :: agent;    Res :: resource;
  It :: item;     M,N :: nnInteger.

:- objects
  buyer,seller        :: agent;
  money               :: resource;
  0..maxInt           :: nnInteger.

:- constants
  price(item)         :: nnInteger;
  has(agent,resource)
           :: additiveFluent(nnInteger);
  buy(item,nnInteger) :: exogenousAction.

buy(It,N) increments has(buyer,It) by N.
buy(It,N) decrements has(seller,It) by N.
buy(It,N) increments has(seller,money)
   by M*N if price(It)=M
                      where M*N =< maxInt.
buy(It,N) decrements has(buyer,money)
   by M*N if price(It)=M
                      where M*N =< maxInt.
nonexecutable buy(It,M) & buy(It,N)
                      where M\=N.
```

Figure 5: Buying and selling

```
:- objects
  newspaper,magazine  :: item.

price(newspaper)=1.  price(magazine)=3.

:- query
maxstep :: 1;
0: has(buyer,money)=6,
   buy(newspaper,2),  buy(magazine,1).
```

Figure 6: Do I have enough cash?

has \$6, and whose label includes buying 2 newspapers and 1 magazine. CCALC responds to this query by finding such an edge.[2] Its reply to a similar question about 1 newspaper and 2 magazines is negative.

7 Reasoning about Motion

Some additive fluents mentioned in the introduction—for instance, the velocity of a particle—are real-valued, rather than integer-valued. CCALC cannot deal with real numbers yet, and its input language does not allow us to express properties of such fluents.

But let's imagine a movable object that is immune to this complication—the spacecraft Integer. Far away from stars and planets, the Integer is not affected by any external forces. As its proud name suggests, the mass of the spacecraft is an integer. For every integer t, the coordinates and all three components of the Integer's velocity vector at time t are integers; the forces applied to the spacecraft by its jet engines over the interval $(t, t+1)$, for any integer t, are constant vectors whose components are integers as well. If the crew of the Integer attempts to violate any of these conditions, the jets fail to operate!

Our formalization of the motion of the Integer uses the fluents $Pos(a)$, where a ranges over $\{X, Y, Z\}$, to represent the current position of the Integer along the a axis. The additive fluents $Vel(a)$ are the components of its velocity. According to Newton's Second Law, the acceleration created by firing a jet can be computed by dividing the force by the mass of the spacecraft. This relationship can be expressed without mentioning the acceleration explicitly—in terms of the change in the velocity over a unit time interval:

$$Fire(j) \textbf{ increments } Vel(a) \textbf{ by } n/Mass \textbf{ if } Force(j, a) = n.$$

To test our representation, we instructed CCALC to answer the following question:

> The mass of the Integer is 1. The Integer has two jets, and the force that can be applied by each jet along each axis is at most 2. The current position of the Integer is $(-1, 0, 1)$, and its current velocity is $(0, 1, 1)$. How can it get to $(0, 3, 1)$ within 1 time unit?

One of the solutions found by CCALC is to apply the forces $(2, 2, 0)$ and $(0, 2, -2)$.

8 Missionaries and Cannibals with Two Boats

In the Missionaries and Cannibals Problem (MCP), three missionaries and three cannibals come to a river and find a boat that can hold two people. If the cannibals ever outnumber the missionaries on either bank, the missionaries will be eaten. How shall they cross? The shortest solution involves 11 steps.

Lifschitz [2000] showed how to express this puzzle and some of its elaborations due to McCarthy [1999] in the language of CCALC. Some simple elaborations of MCP in the spirit of this work require that the number of members of a group (missionaries or cannibals) at a given location be treated as an additive fluent. This is the case, for instance, when several boats are available and are allowed to operate concurrently.

Using the ideas of this paper, we formalized the modification of MCP in which the travelers find two boats: a small boat that holds one, and a bigger boat that holds two. Using our formalization, CCALC has determined that the modified problem can be solved in 7 steps.

9 Properties of Additive Constants

By examining Figure 3 in isolation from its symbolic description in Figure 2 we can see that the constant *InStore* exhibits some features typical for additive fluent constants.

Consider, for instance, the edges that start at the vertex $InStore = 2$ and are labeled by the events $\{Buy(A)\}$ and

[2] This example involves the concurrent execution of two actions, but in general the CCALC implementation of additive fluents does not impose any specific restriction on the number of actions that can be executed concurrently.

$\{Buy(B)\}$. Each of them leads to the vertex $InStore = 1$, so that each of these two events, when it occurs in the state $InStore = 2$, increments the value of *InStore* by -1. In accordance with the intuitive idea of an additive fluent, we can expect that the "union" of these events, when it occurs in the same state, will increment the value of *InStore* by $(-1) + (-1)$. And this is true, because the edge in Figure 3 that starts at the vertex $InStore = 2$ and is labeled $\{Buy(A), Buy(B)\}$ leads to the vertex $InStore = 0$.

Proposition 1 below generalizes this observation to a class of action descriptions in the language $\mathcal{C}+$ extended as described in Sections 4, 5. By D we denote any action description in this language.

About events e_0, e_1,...,e_n ($n \geq 0$) in the transition system represented by D we say that e_0 is a *disjoint union* of e_1,...,e_n if, for every Boolean action constant a,

- if $e_0(a) = \mathrm{t}$ then there exists a unique $i > 0$ such that $e_i(a) = \mathrm{t}$; for this i, $e_0(a') = e_i(a')$ for every non-Boolean action constant a';
- if $e_0(a) = \mathrm{f}$ then, for all $i > 0$, $e_i(a) = \mathrm{f}$.

In the rest of this section we assume that the set of numbers is a commutative group.

Proposition 1 *Let c be an additive fluent constant, let s, $s_0, \ldots, s_n$ ($n \geq 0$) be states, and let $e_0, \ldots, e_n$ be events such that e_0 is a disjoint union of $e_1, \ldots, e_n$. If, for all $i \in \{0, \ldots, n\}$, the transition system represented by D contains an edge that leads from s to s_i and is labeled e_i then*

$$s_0(c) - s(c) = \sum_{i=1}^{n} (s_i(c) - s(c)).$$

The special case corresponding to $n = 0$ tells us that additive fluent constants are not affected by "trivial" events. In this sense, they are similar to the fluent constants for which inertia is postulated:

Corollary 1 *Let e be an event such that for every Boolean action constant a, $e(a) = \mathrm{f}$. If the transition system represented by D contains an edge that leads from a state s to a state s' and is labeled e then, for any additive fluent constant c, $s'(c) = s(c)$.*

The special case corresponding to $n = 1$ implies that the effects of any set of actions on an additive fluent is deterministic:

Corollary 2 *If the transition system represented by D contains an edge that leads from a state s to a state s_0 and is labeled e, and an edge that leads from s to a state s_1 and is also labeled e, then, for any additive fluent constant c, $s_0(c) = s_1(c)$.*

10 Discussion

In this paper we showed how an implemented, declarative language for describing actions can be used to talk about the effects of actions on additive fluents. This was accomplished by extending the syntax of the action language $\mathcal{C}+$ from [Giunchiglia *et al.*, 2003] by **increment** laws and by showing how to treat these laws as abbreviations.

It is interesting to note that this treatment of additive fluents would have been impossible if, instead of $\mathcal{C}+$, we used its predecessor $\mathcal{C}$ from [Giunchiglia and Lifschitz, 1998]. Non-Boolean, non-exogenous action constants such as $Contr(a, c)$, and action dynamic laws such as (10) and (11) are among the features of $\mathcal{C}+$ that were not available in $\mathcal{C}$.

In literature on planning, fluents with numerical values are often referred to as "resources" [Koehler, 1998]. The concurrent execution of the actions that involve resources is usually limited to the "serializable" case, when all ways of sequencing the concurrent actions are well-defined and equivalent. This condition is not satisfied, however, for many uses of additive fluents, including the space travel example (Section 7) and the modified MCP (Section 8). Consider, for instance, the state with two cannibals and three missionaries on the first bank, and imagine that the missionaries are using both boats to cross simultaneously. The concurrent execution of the actions cannot be replaced by the larger boat crossing first and the smaller boat crossing after that—in the state between the two actions, the missionaries on the first bank would be outnumbered by the cannibals. This example shows that crossing the river in the modified MCP is not serializable.

Acknowledgements

We are grateful to Yuliya Babovich, Jonathan Campbell, Esra Erdem, Selim Erdoğan, Paolo Ferraris and Wanwan Ren for useful discussions related to the subject of this paper. This work was partially supported by the National Science Foundation under Grant IIS-9732744 and by the Texas Higher Education Coordinating Board under Grant 003658-0322-2001.

References

[Gelfond and Lifschitz, 1998] Michael Gelfond and Vladimir Lifschitz. Action languages.[3] *Electronic Transactions on AI*, 3:195–210, 1998.

[Giunchiglia and Lifschitz, 1998] Enrico Giunchiglia and Vladimir Lifschitz. An action language based on causal explanation: Preliminary report. In *Proc. AAAI-98*, pages 623–630. AAAI Press, 1998.

[Giunchiglia *et al.*, 2003] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories.[4] *Artificial Intelligence*, 2003. To appear.

[Koehler, 1998] Jana Koehler. Planning under resource constraints. In *Proc. ECAI-98*, pages 489–493, 1998.

[Lifschitz, 2000] Vladimir Lifschitz. Missionaries and cannibals in the Causal Calculator. In *Principles of Knowledge Representation and Reasoning: Proc. Seventh Int'l Conf.*, pages 85–96, 2000.

[McCarthy, 1999] John McCarthy. Elaboration tolerance.[5] In progress, 1999.

[3] `http://www.ep.liu.se/ea/cis/1998/016/` .

[4] `http://www.cs.utexas.edu/users/vl/papers/nmct.ps` .

[5] `http://www-formal.stanford.edu/jmc/elaboration.html` .