The Concurrent, Continuous Fluent Calculus

MICHAEL THIELSCHER
Department of Computer Science, Dresden University of Technology, 01062 Dresden (Germany)
e-mail: mit@inf.tu-dresden.de

Abstract. The Fluent Calculus belongs to the established predicate calculus formalisms for reasoning about actions. Its underlying concept of state update axioms provides a solution to the basic representational and inferential Frame Problems in pure first-order logic. Extending a recent research result, we present a Fluent Calculus to reason about domains involving continuous change and where actions occur concurrently.

0.1 Introduction

Research into Cognitive Robotics aims at explaining and modeling intelligent acting in a dynamic world. Whenever intelligent behavior is understood as resulting from correct reasoning on correct representations, the classical Frame Problem [6] is a fundamental theoretical challenge: Given a representation of the effects of the available actions, how can one formally capture a crucial regularity of the real world, namely, that an action usually does not have arbitrary other effects? Explicitly specifying for each single potential effect that it is actually not an effect of a particular action, is obviously unsatisfactory both as a representation technique and as regards efficient inferencing [2]. The predicate calculus formalism of the Fluent Calculus [14], which roots in the logic programming approach of [5], provides a basic solution to both the representational and the inferential aspect of the Frame Problem. The simple Fluent Calculus has been extended into various directions, including nondeterministic actions [15], ramifications [12], and sensing actions [16].

In [13], we have developed a theory based on the Fluent Calculus of reasoning about actions and planning in domains involving continuous change. In being applicable to planning problems with incomplete knowledge of world states, this approach has been shown to provide a better generalization of Green's classical definition of planning by deduction [4] than the Situation Calculus-based formalism of [8]. The latter, on the other hand, supports

concurrent actions while [13] applies only to non-concurrent worlds.

In this paper, we extend our theory so as to obtain a Fluent Calculus which allows for specifying and reasoning about domains involving continuous change and where actions occur concurrently. With the resulting axiomatization technique we will be able to solve planning problems of the following kind: Suppose you want to boil an egg for exactly 15 minutes. The only available tool for measuring time are two sandglasses, one of which runs for 7 minutes while the other one runs for 11 minutes. This problem involves continuous change and requires to turn the two sandglasses simultaneously at some point. Which sequence of actions achieves the goal?[1]

0.2 Fluents, Processes, and States

The Fluent Calculus is a sorted language of classical logic with equality. There are four standard sorts, namely, FLUENT, STATE (of which FLUENT is a sub-sort), ACTION, and SIT. For the concurrent, continuous Fluent Calculus, we add the two sorts CONCURRENT (of which ACTION is a sub-sort) and REAL, to be interpreted as the real numbers. The latter is accompanied by the usual arithmetic operations along with their standard interpretation.

For our example planning problem, we additionally introduce the domain sort SANDGLASS, with just two elements:[2]

$$g = G_7 \vee g = G_{11} \tag{1}$$

along with the specification of their sand load,

$$\mathit{Capacity}(G_7) = 7 \wedge \mathit{Capacity}(G_{11}) = 11 \tag{2}$$

The so-called fluents are the basic entities to describe states of a dynamic system. Each fluent represents an atomic property that may be affected by actions and hence change in the course of time. In the continuous Fluent

[1] If the reader considers this instance too simple, try the problem of measuring 9 minutes with two sandglasses running for 4 and 7 minutes, respectively.

[2] A word on the notation: Predicate and function symbols, including constants, start with a capital letter whereas variables are in lower case, sometimes with sub- or superscripts. Free variables in formulas are assumed universally quantified. Throughout the paper, fluent variables are denoted by the letter f, state variables by the letter z, action variables by the letter a, concurrency variables by the letter c, situation variables by the letter s, real-valued variables by the letters d and t, and sandglass variables by the letter g, all possibly with sub- or superscript.

Calculus, a fluent can represent a complex continuous process, such as the constant flow of sand through a sandglass. The various states in our example domain will be modeled on the basis of the following four fluents:

$$\begin{aligned} \mathit{Idle} &: \textsc{sandglass} \mapsto \textsc{fluent} \\ \mathit{Running} &: \textsc{sandglass} \times \textsc{real} \times \textsc{real} \mapsto \textsc{fluent} \\ \mathit{Boiling} &: \textsc{real} \mapsto \textsc{fluent} \\ \mathit{Boiled} &: \textsc{real} \mapsto \textsc{fluent} \end{aligned}$$

Fluent $\mathit{Idle}(g)$ indicates that sandglass g is idling; fluent $\mathit{Running}(g, d, t)$ indicates the process of sandglass g running, having started at time t with an amount of sand in the top that lasts for exactly d minutes;[3] fluent $\mathit{Boiling}(t)$ says that the egg is being boiled, having started at time t; and fluent $\mathit{Boiled}(t)$ holds if the egg has been taken out of the pot and has been boiled for t minutes. In addition to domain fluents, the continuous Fluent Calculus includes one pre-defined fluent, $\mathit{StartTime} : \textsc{real} \mapsto \textsc{fluent}$, an instance of which in a state indicates the time at which the system has entered that state. Adopting a notation from [1], we introduce the following axioms of uniqueness of names for our example domain:

$$\mathit{UNA}[G_7, G_{11}] \wedge \mathit{UNA}[\mathit{Idle}, \mathit{Running}, \mathit{Boiling}, \mathit{Boiled}, \mathit{StartTime}] \qquad (3)$$

The distinctive feature of the Fluent Calculus in comparison with the Situation Calculus of [7], or the Event Calculus of [10], is its explicit, abstract notion of a state besides that of a situation.[4] We have said that FLUENT is a sub-sort of STATE. Each single fluent represents the particular state in which just this fluent holds and nothing else is true. State terms can be joined together by the binary function " $\circ$ " of sort $\textsc{state} \times \textsc{state} \mapsto \textsc{state}$, denoting the state in which precisely the elements of both arguments hold. We write this function symbol in infix notation. In addition, the special constant $\emptyset$ of sort STATE denotes the state in which no fluents at all hold.

Fundamental axioms for states stipulate crucial properties of the connection function " $\circ$ ". Firstly, it is an associative-commutative operation and has $\emptyset$ as unit element, so that states are equal if they differ only in the order

[3] For example, if G_7 is turned upside down at time $t = 7$ and again at time $t = 11$, then the fluent $\mathit{Running}(G_7, 4, 11)$ will hold afterwards (for at most 4 minutes).

[4] To clarify terminology, a situation is characterized by a sequence of actions whereas a state is characterized by the fluents that hold.

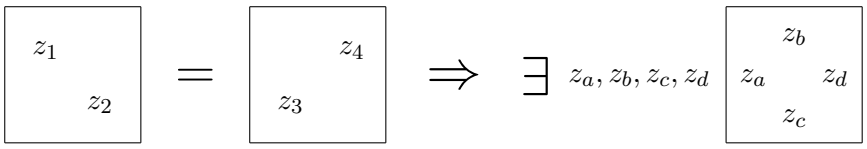

Figure 1: The Levi axiom:[5] If some state (symbolized by a square) can be partitioned into z_1, z_2 as well as into z_3, z_4, then it can be partitioned into z_a, z_b, z_c, z_d as depicted.[6]

in which their elements are composed together:

$$\begin{aligned} (z_1 \circ z_2) \circ z_3 &= z_1 \circ (z_2 \circ z_3) \\ z_1 \circ z_2 &= z_2 \circ z_1 \\ z \circ \emptyset &= z \end{aligned} \tag{F1}$$

(The law of associativity allows us to omit parentheses in nested applications of "$\circ$".) Secondly, the following axiom states that each fluent is a non-empty, irreducible state, and so represents a state in which just this fluent holds:

$$z = f \supset z \neq \emptyset \wedge [\, z = z' \circ z'' \supset z' = \emptyset \vee z'' = \emptyset \,] \tag{F2}$$

Finally, Levi's axiom is needed to conclude inequality of two state terms if they do not contain the same fluents:

$$\begin{gathered} z_1 \circ z_2 = z_3 \circ z_4 \supset \\ (\exists z_a, z_b, z_c, z_d)\,(z_1 = z_a \circ z_b \wedge z_2 = z_c \circ z_d \wedge z_3 = z_a \circ z_c \wedge z_4 = z_b \circ z_d) \end{gathered} \tag{F3}$$

Figure 1 gives an illustrative graphical interpretation of this axiom.

The conjunction of these foundational axioms along with a set of domain-dependent unique names-axioms *UNA*, like, e.g., (3), is denoted by *EUNA*. A logical consequence of these axioms are, as has been shown in [11], the following two laws, which are of great practical value when it comes to calculating with state equations.

[5] The naming comes from a lemma in trace theory [3].

[6] It should be noted, however, that the picture is slightly misleading: In case z_1, z_2, z_3, z_4 contain multiple occurrences of sub-terms, the states z_a, z_b, z_c, z_d are not necessarily uniquely determined, as the reader may verify with the example $(f \circ f) \circ (f \circ f) = f \circ (f \circ f \circ f)$.

Proposition 1 (Cancellation Law) *In all models of EUNA we have*

$$f \circ z = f \circ z' \supset z = z'$$

Proposition 2 (Distribution Law) *In all models of EUNA we have*

$$f_1 \neq f_2 \supset f_1 \circ z_1 = f_2 \circ z_2 \supset (\exists z')\, z_1 = f_2 \circ z' \wedge (\exists z')\, z_2 = f_1 \circ z'$$

For example, given (3) the state equation $Idle(G_{11}) \circ Running(G_7, d, t) = Running(g, 5, t_0) \circ z$ can be simplified, with the help of these two laws, to $g = G_7,\ d = 5,\ t = t_0$, and $z = Idle(G_{11})$.

Based on the concept of state terms, the function $State : \text{SIT} \mapsto \text{STATE}$ relates a situation to the state of the world in that situation. It is, however, never assumed that world states can be completely specified. Rather, the abstract term $State(\sigma)$ may be constrained by means of equations and inequations specifying what is known about situation σ. As an example, let the constant S_0 be a denotation of the initial situation, then the following formula specifies some assumptions about the initial state for our planning problem, namely, that the start time is 0, the two sandglasses idle, and the egg is neither boiling nor boiled:

$$\begin{aligned}(\exists z)\,(&State(S_0) = Idle(G_7) \circ Idle(G_{11}) \circ StartTime(0) \circ z \\ &\wedge (\forall t, z')\, z \neq Boiling(t) \circ z' \ \wedge\ (\forall t, z')\, z \neq Boiled(t) \circ z'\,)\end{aligned} \tag{4}$$

Put in words, of $State(S_0)$ we assume that it includes each of $Idle(G_7)$, $Idle(G_{11})$, and $StartTime(0)$. Arbitrary other fluents z may hold, too, except for $Boiling(t)$ and $Boiled(t)$, which are thus specified to be false in S_0 for any t.

Fluent Calculus specifications frequently use the expression $Holds(f, z)$ along with the common $Holds(f, s)$—stating that fluent f holds in state z and situation s, respectively—, though they are not part of the signature but mere abbreviations of equality sentences:

$$\begin{aligned}Holds(f, z) &\stackrel{\text{def}}{=} (\exists z')\, z = f \circ z' \\ Holds(f, s) &\stackrel{\text{def}}{=} Holds(f, State(s))\end{aligned}$$

With these macros at hand the initial situation in our planning problem could have equally well been specified as follows:

$$\begin{aligned}&Holds(Idle(G_7), S_0) \wedge Holds(Idle(G_{11}), S_0) \wedge Holds(StartTime(0), S_0) \wedge \\ &(\forall t)\,(\neg Holds(Boiling(t), S_0) \wedge \neg Holds(Boiled(t), S_0))\end{aligned}$$

The *Holds* expressions also help increasing the readability of specifications like domain-dependent state constraints, which are formulas that are supposed to hold in all states which can occur in reality. To formally characterize these so-called consistent states, we use the predicate $Cons : \textsc{state}$. By the following foundational axiom, all states associated with a situation are required to be consistent:

$$Cons(State(s)) \tag{F4}$$

Our example domain, for instance, exhibits the following constraints, which state, respectively, that a sandglass is running just in case it is not idling; that a sandglass cannot be running in two different ways in the same situation; that a sandglass cannot be running with no sand in its top nor with more than what its capacity allows; that the egg cannot both boil and already being boiled in the same situation; and that the egg cannot be boiling nor being boiled for two different time spans:

$$\begin{aligned}
&Cons(z) \supset (\exists d, t_0)\, Holds(Running(g, d, t_0), z) \equiv \neg Holds(Idle(g), z)\\
&Cons(z) \supset Holds(Running(g, d, t_0), z) \wedge Holds(Running(g, d', t_0'), z) \supset\\
&\qquad\qquad t_0 = t_0' \wedge d = d'\\
&Cons(z) \supset Holds(Running(g, d, t_0), z) \supset 0 < d \leq Capacity(g)\\
&Cons(z) \supset \neg[Holds(Boiling(t), z) \wedge Holds(Boiled(t'), z)]\\
&Cons(z) \supset Holds(Boiling(t), z) \wedge Holds(Boiling(t'), z) \supset t = t'\\
&Cons(z) \supset Holds(Boiled(t), z) \wedge Holds(Boiled(t'), z) \supset t = t'
\end{aligned} \tag{5}$$

In addition, a foundational state constraint says that multiple occurrences of fluents are prohibited:

$$Cons(z) \supset (\forall f, z')\, z \neq f \circ f \circ z' \tag{F5}$$

(It will be explained in the following section why " $\circ$ " is not required to be idempotent instead.) The continuous Fluent Calculus includes one further foundational state constraint, which stipulates that the starting time of a state associated with a situation be unique:

$$Cons(z) \supset (\exists! t)\, Holds(StartTime(t), z) \tag{F6}$$

The standard function $Start : \textsc{state} \mapsto \textsc{real}$ denotes the starting time of a state by referring to the fluent *StartTime*, provided it is unique:

$$\begin{aligned}
&(\exists! t)\, Holds(StartTime(t), z) \supset\\
&\quad (\forall t)\, (Holds(StartTime(t), z) \supset Start(z) = t)
\end{aligned} \tag{F7}$$

0.3 Actions, Concurrency, and Successor States

States change as the result of the performance of actions. In continuous worlds, a distinction is usually made between deliberative actions and so-called natural ones, which happen 'automatically' [9, 8]. For our example we shall use the following four actions, the second of which is a natural one:

$$\begin{aligned} \mathit{Turn} &: \textsc{sandglass} \mapsto \textsc{action} \\ \mathit{Stops} &: \textsc{sandglass} \mapsto \textsc{action} \\ \mathit{StartBoiling} &: \mapsto \textsc{action} \\ \mathit{EndBoiling} &: \mapsto \textsc{action} \end{aligned}$$

Action $\mathit{Turn}(g)$ denotes turning upside down sandglass g; natural action $\mathit{Stops}(g)$ denotes automatic termination of sandglass g running; and actions *StartBoiling* and *EndBoiling* denote, respectively, the action of starting and ending the boiling of the egg. The following closure axiom says that these are all actions in our domain:

$$\begin{aligned} &(\exists g)\, a = \mathit{Turn}(g) \;\vee\; (\exists g)\, a = \mathit{Stops}(g) \\ &\vee\; a = \mathit{StartBoiling} \;\vee\; a = \mathit{EndBoiling} \end{aligned} \tag{6}$$

In order to represent the concurrent performance of actions, single action terms may be composed to terms of sort CONCURRENT by a binary function. The latter is denoted by " $\cdot$ " and written in infix notation. This function, which is of type $\textsc{concurrent} \times \textsc{concurrent} \mapsto \textsc{concurrent}$, is assumed to be an associative-commutative operation with unit element ϵ (read: *no-op*) of sort CONCURRENT. The foundational axioms for concurrent action terms are analogous to those for state terms:

$$\begin{aligned} (c_1 \cdot c_2) \cdot c_3 &= c_1 \cdot (c_2 \cdot c_3) \\ c_1 \cdot c_2 &= c_2 \cdot c_1 \\ c \cdot \epsilon &= c \end{aligned} \tag{F8}$$

$$c = a \;\supset\; c \neq \epsilon \wedge [\, c = c' \cdot c'' \;\supset\; c' = \epsilon \vee c'' = \epsilon \,] \tag{F9}$$

$$\begin{aligned} &c_1 \cdot c_2 = c_3 \cdot c_4 \;\supset \\ &(\exists c_a, c_b, c_c, c_d)\, (c_1 = c_a \cdot c_b \wedge c_2 = c_c \cdot c_d \wedge c_3 = c_a \cdot c_c \wedge c_4 = c_b \cdot c_d) \end{aligned} \tag{F10}$$

In addition, the following second-order closure axiom restricts concurrent actions to finite collections of single actions:

$$(\forall \Pi)\, \{\, \Pi(\epsilon) \wedge (\forall a, c)\, (\Pi(c) \supset \Pi(a \cdot c)) \;\supset\; (\forall c)\, \Pi(c) \,\} \tag{F11}$$

Similar to the *Holds* macro we use the abbreviation $In(c_1, c)$ to denote that concurrent action c_1 is included in concurrent action c:

$$In(c_1, c) \stackrel{\text{def}}{=} (\exists c')\, c = c_1 \cdot c'$$

Preconditions of actions are specified using the standard predicate $Poss$: CONCURRENT × REAL × STATE, meaning that a (concurrent) action is possible at a certain time in a certain state. A standard way of defining preconditions in concurrent domains is to first give preconditions separately for each single action and, then, to define the possibility of concurrent executions. In our example planning domain, suitable precondition axioms for the single actions are the following ones.

A sandglass automatically stops running as soon as it ran out of sand:

$$Poss(Stops(g), t, z) \equiv (\exists d, t_0)\,(Holds(Running(g, d, t_0), z) \wedge t = t_0 + d) \quad (7)$$

In order to be able to keep track of the time that has passed, turning a sandglass g upside down shall be possible only at the beginning, where we have set the time to 0, or when some sandglass (which could be g itself) stops running:

$$Poss(Turn(g), t, z) \equiv t = 0 \wedge t \geq Start(z) \vee (\exists g')\, Poss(Stops(g'), t, z) \quad (8)$$

Likewise, starting to boil the egg shall be possible only at the very beginning or when a sandglass has just stopped; further preconditions are that the egg is not currently being boiled nor boiled already:

$$\begin{aligned} &Poss(StartBoiling, t, z) \equiv \\ &\quad [\, t = 0 \wedge t \geq Start(z) \vee (\exists g)\, Poss(Stops(g), t, z)\,] \\ &\quad \wedge \neg(\exists t_0)\,(Holds(Boiling(t_0), z) \vee Holds(Boiled(t_0), z)) \end{aligned} \quad (9)$$

Finally, boiling can be brought to an end at any point in time measured by the stopping of a sandglass:

$$\begin{aligned} &Poss(EndBoiling, t, z) \equiv \\ &\quad (\exists t_0)\, Holds(Boiling(t_0), z) \wedge (\exists g)\, Poss(Stops(g), t, z) \end{aligned} \quad (10)$$

A standard requirement for the possibility to perform a concurrent action c is that it is not empty, that each involved single action be possible at that time, and that no action term occurs twice in c.[7] Additional constraints may state that two or more actions are in mutual conflict. No such

[7] The last condition is similar to foundational axiom (F5). Requiring " $\cdot$ " to be idempotent instead would not do, as will become clear shortly.

restriction applies to our example domain, hence the following precondition axiom for the concurrent performance of actions:

$$Poss(c,t,z) \;\equiv\; c \neq \epsilon \wedge (\forall a)\,(In(a,c) \supset Poss(a,t,z)) \wedge \neg(\exists a)\,In(a{\cdot}a,c) \quad (11)$$

In the simple Fluent Calculus, effects of actions are specified by means of so-called state update axioms, whose core is an equation relating a state after the performance of an action to the state prior to it. These equations do not mention non-effects, which is how the Fluent Calculus solves the representational Frame Problem. Moreover, one equation always describes the entire change caused by an action, so that a single state equation suffices to infer the result of an action. This is how the Fluent Calculus solves the inferential Frame Problem. An extension of this basic solution is provided by what shall be called recursive state update axioms, which allow for specifying the effect of an action *relative* to the effect of arbitrary other actions performed concurrently. Let $Succ : \text{CONCURRENT} \times \text{REAL} \times \text{STATE} \mapsto \text{STATE}$ denote the successor state of performing a (concurrent) action at a certain time in a certain state, then this is the general form of recursive state update axioms:

$$\Delta(t,z) \;\supset\; (\exists \vec{y})Succ(\alpha(\vec{x}) \cdot c, t, z) \circ \vartheta^- = Succ(c,t,z) \circ \vartheta^+$$

Here, $\Delta(t,z)$ is a first-order formula specifying conditions on execution time t and state z for the update equation to apply; $\vec{y}$ are the variables which occur in ϑ^-, ϑ^+ but not in $\vec{x}$; and the two terms ϑ^- and ϑ^+ are the *additional* negative and positive, respectively, effects which occur if α is performed *besides* c.[8]

Consider the action of starting to boil the egg at time t. Its only effect is to initiate the $Boiling(t)$ fluent:

$$\begin{array}{l} Poss(StartBoiling \cdot c, t, z) \;\supset \\ \quad Succ(StartBoiling \cdot c, t, z) = Succ(c,t,z) \circ Boiling(t) \end{array} \quad (12)$$

The action of ending the boiling of the egg at some time t has the negative effect of terminating fluent $Boiling(t_0)$, where t_0 is the time at which

[8] This way of representing negative effects is the reason for not stipulating that " $\circ$ " be idempotent, contrary to what one might intuitively expect. For if the function were idempotent, then the equation in the update axiom would not imply that each fluent in ϑ^- be false in $Succ(\alpha(\vec{x}) \cdot c, t, z)$. Likewise, if function " $\cdot$ " were idempotent, then $\alpha(\vec{x}) \cdot c$ could equal c, in which case the update axioms gave rise to a circular equation, which is likely to be inconsistent.

boiling has started, and of initiating fluent $Boiled(t')$, where t' denotes the amount of time the egg has been boiled. Obviously, $t' = t - t_0$; hence the following state update axiom:

$$\begin{aligned} &Poss(EndBoiling \cdot c, t, z) \supset \\ &\quad (\exists t_0)\, Succ(EndBoiling \cdot c, t, z) \circ Boiling(t_0) = \\ &\qquad\qquad Succ(c, t, z) \circ Boiled(t - t_0) \end{aligned} \tag{13}$$

On the basis of recursive state update axioms, the overall effect of a particular concurrent action $\alpha_1 \cdot \ldots \cdot \alpha_n$ $(n \geq 1)$ performed at some time τ in some state ζ is determined by a set of recursive equations, which are obtained as the consequents of instances of the appropriate update axioms:

$$\begin{aligned} Succ(\alpha_1 \cdot \alpha_2 \cdot \alpha_3 \cdot \ldots \cdot \alpha_n, \tau, \zeta) \circ \vartheta_1^- &= Succ(\alpha_2 \cdot \alpha_3 \ldots \cdot \alpha_n, \tau, \zeta) \circ \vartheta_1^+ \\ Succ(\alpha_2 \cdot \alpha_3 \cdot \ldots \cdot \alpha_n, \tau, \zeta) \circ \vartheta_2^- &= Succ(\alpha_3 \cdot \ldots \cdot \alpha_n, \tau, \zeta) \circ \vartheta_2^+ \\ &\vdots \\ Succ(\alpha_n, \tau, \zeta) \circ \vartheta_n^- &= Succ(\epsilon, \tau, \zeta) \circ \vartheta_n^+ \end{aligned}$$

With the help of these equations, the result of the concurrent action is inferred by decomposing the latter and successively inferring the effects of the components. Any such chain of inference steps ends in the base case defined by the following foundational axiom, which states that the only effect of the empty concurrent action at time t is to give rise to a new starting time:

$$(\exists t')\, Succ(\epsilon, t, z) \circ StartTime(t') = z \circ StartTime(t) \tag{F12}$$

Inferring the cumulative effect via separate state update axioms allows one to specify effects of actions in a modular way, that is, separately for each single action. At the same time, non-effects of concurrent actions are carried through each equational inference step and so need no extra axioms to be inferred as such. This is how the approach generalizes the solution to the representational and inferential Frame Problem from a single one to a whole chain of state update axioms for concurrency.

Concurrently performed actions may not be independent, in which case the combined effect is not a mere accumulation. To account for such exceptions, a recursive state update axiom may be qualified using the reserved predicate $Affects : \textsc{concurrent} \times \textsc{concurrent}$. An instance $Affects(c, c')$ shall indicate that if c is performed concurrently with c' at time t in state z, then the usual effects of c' will not materialize. Our example domain exhibits one such case of interference: Whenever a sandglass is turned

upside down at the very time it stops running, the $Stops(g)$ action will not produce its usual effect (namely, the idling of g).

We therefore first define

$$Affects(c, c') \equiv (\exists g)\,(In(Turn(g), c) \wedge In(Stops(g), c')) \tag{14}$$

On this basis, we can specify the effect, upon non-cancellation, of a sandglass to stop running—the run is terminated and the sandglass starts to idle:

$$\begin{aligned} &Poss(Stops(g) \cdot c, t, z) \wedge \neg Affects(c, Stops(g)) \supset \\ &(\exists d, t_0)\, Succ(Stops(g) \cdot c, t, z) \circ Running(g, d, t_0) = \\ &\qquad\qquad Succ(c, t, z) \circ Idle(g) \end{aligned} \tag{15}$$

For the action of turning a sandglass upside down, we have to distinguish three cases. First, the sandglass may be idling at the time the action is performed, in which case the idling is terminated and the sandglass starts running with maximal load:

$$\begin{aligned} &Poss(Turn(g) \cdot c, t, z) \wedge Holds(Idle(g), z) \supset \\ &\qquad Succ(Turn(g) \cdot c, t, z) \circ Idle(g) = \\ &\qquad\quad Succ(c, t, z) \circ Running(g, Capacity(g), t) \end{aligned} \tag{16}$$

Second, if the sandglass is turned upside down while running and it is not about to stop at the same time, then the $Running$ fluent is replaced by a modified one. The expected runtime of the new run is calculated as the difference between the capacity and what is still in the upper bowl of the sandglass at the time of turning:

$$\begin{aligned} &Poss(Turn(g) \cdot c, t, z) \wedge (\exists d, t_0)\, Holds(Running(g, d, t_0), z) \\ &\wedge \neg Affects(Turn(g), c) \\ &\supset (\exists d, t_0)\, Succ(Turn(g) \cdot c, t, z) \circ Running(g, d, t_0) = \\ &\qquad Succ(c, t, z) \circ Running(g, Capacity(g) - d + t - t_0, t) \end{aligned} \tag{17}$$

Third, if turning and stopping happen at the very same time, then the combined effect of both actions is that the current run is terminated and a new one is initiated:

$$\begin{aligned} &Poss(Turn(g) \cdot Stops(g) \cdot c, t, z) \supset \\ &\quad (\exists d, t_0)\, Succ(Turn(g) \cdot Stops(g) \cdot c, t, z) \circ Running(g, d, t_0) = \\ &\qquad Succ(c, t, z) \circ Running(g, Capacity(g) - d + t - t_0, t) \end{aligned} \tag{18}$$

The reader may notice the difference to the preceding axiom: Here, the concurrent $Stops$ action is processed, too, hence no update axiom for this

action will apply during the recursion for the remaining actions c. In this way we avoid calculating the usual effect of *Stops*, namely, to introduce idling (cf. (15)).

Recall, for example, the specification of the initial state in our planning problem, (4), and consider the concurrent action $C_1 = Turn(G_7) \cdot Turn(G_{11}) \cdot StartBoiling$ to be performed at time 0. From (4) and the relevant precondition axioms, (8), (9), and (11), it follows that $Poss(C_1, 0, State(S_0))$. Drawing appropriate instances of the state update axioms (16) and (12), after evaluating the condition parts of these axioms we obtain this set of recursive state equations (with the last equation being the appropriate instance of foundational axiom (F12)):

$$Succ(Turn(G_7) \cdot Turn(G_{11}) \cdot StartBoiling, 0, State(S_0)) \circ Idle(G_7) = Succ(Turn(G_{11}) \cdot StartBoiling, 0, State(S_0)) \circ Running(G_7, 7, 0)$$

$$Succ(Turn(G_{11}) \cdot StartBoiling, 0, State(S_0)) \circ Idle(G_{11}) = Succ(StartBoiling, 0, State(S_0)) \circ Running(G_{11}, 11, 0)$$

$$Succ(StartBoiling, 0, State(S_0)) = Succ(\epsilon, 0, State(S_0)) \circ Boiling(0)$$

$$(\exists t')\, Succ(\epsilon, 0, State(S_0)) \circ StartTime(t') = State(S_0) \circ StartTime(0)$$

Given (4), these equations entail, applying the cancellation and distribution laws of Propositions 1 and 2 and repeating the relevant negative information about z,

$$(\exists z)\,(Succ(C_1, 0, State(S_0)) = z \circ StartTime(0) \circ Boiling(0) \circ Running(G_{11}, 11, 0) \circ Running(G_7, 7, 0) \wedge (\forall t, z')\, z \neq Boiled(t) \circ z')$$

From (7) it then follows that $Poss(Stops(G_7), 7, Succ(C_1, 0, State(S_0)))$; and the reader may verify that if, say, $C_2 = Stops(G_7) \cdot Turn(G_7)$, then we can conclude

$$(\exists z)\,(Succ(C_2, 7, Succ(C_1, 0, State(S_0))) = z \circ StartTime(7) \circ Boiling(0) \circ Running(G_{11}, 11, 0) \circ Running(G_7, 7, 7) \wedge (\forall t, z')\, z \neq Boiled(t) \circ z')$$

0.4 Situations, Natural Actions, and Trajectories

In worlds which involve continuous change it is likely that at some point processes terminate automatically, or that two or more ongoing processes

eventually affect each other. To reflect this, the Fluent Calculus for continuous change includes the distinction between deliberative and natural actions. The latter are not subject to the free will of a planning agent. Rather they happen automatically under certain conditions. An example taken from our planning problem is the natural action of a sandglass to stop running. The standard predicate *Natural* of sort ACTION (adopted from [8]) is used to discriminate the actions of a domain which are natural ones, as in

$$Natural(a) \equiv (\exists g)\, a = Stops(g) \tag{19}$$

For a suitable treatment of natural actions, and in particular for modeling the autonomous state updates they cause, the crucial notion underlying the Fluent Calculus for continuous change is that of a situation tree with trajectories [13]. As in the basic Situation Calculus of [7], situations are characterized by sequences of actions performed by an agent. The standard function *Do* of sort CONCURRENT $\times$ REAL $\times$ SIT $\mapsto$ SIT denotes the situation which results from performing a (concurrent) deliberative action at a certain time in a situation. Natural actions, on the other hand, may in any situation cause an autonomous evolution of the state associated with that situation.[9] To this end, each situation has a trajectory. A trajectory is a sequence of states. The world state resulting from a deliberative action is always the first one on a trajectory. The further evolution of that trajectory is determined by the natural actions that are expected to happen. The performance of a deliberative action, on the other hand, brings about another situation again, with its own trajectory; see Figure 2.

The evolution of a trajectory is modeled using the predicate *Trajectory*: STATE $\times$ STATE, an instance $Trajectory(z, z')$ of which indicates that z' occurs on the trajectory rooted in z. The correct evolution is given by considering all natural actions that, if considered in isolation, are expected to happen eventually and then to let those define the next state update which happen next in time [9]. To facilitate the formalization of this principle, we first introduce two macros. The expression $ExpectedNatActions(c, t, z)$ shall indicate that in state z actions c are all the natural actions that are expected to happen at time t:

$$\begin{aligned} ExpectedNatActions(c, t, z) \stackrel{\text{def}}{=}\ & c \neq \epsilon \wedge \neg(\exists a)\, In(a \cdot a, c) \wedge \\ & (\forall a)\, (In(a, c) \equiv Natural(a) \wedge Poss(a, t, z)) \end{aligned}$$

[9] Our theory differs in this respect from the Situation Calculus-based approach of [8], where deliberative and natural actions are intertwined in situation terms.

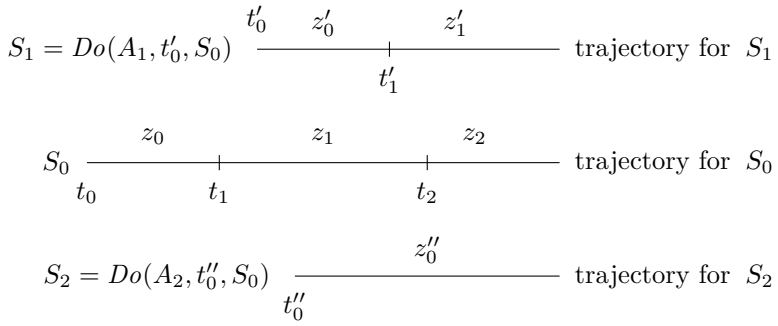

Figure 2: Each situation has its own trajectory, which describes how the state evolves according to the expected natural actions. In the example shown here, at the time t'_0 when the deliberative action A_1 is performed in situation S_0, the world is no longer in the initial state z_0 due to a natural action happening at time $t_1 < t'_0$, which causes state z_1 to arise. The effect of A_1 is to transform z_1 into z'_0, which thus becomes the initial state of the trajectory for situation $Do(A_1, t'_0, S_0)$. If deliberative action A_2 were performed in S_0 at time t''_0, then this gave rise to yet another development of the world, etc.

Recall, for example, state $Z_2 = Succ(C_2, 7, Succ(C_1, 0, State(S_0)))$, which has been shown to equal

$$Running(G_{11}, 11, 0) \circ Running(G_7, 7, 7) \circ Boiling(0) \circ StartTime(7) \circ z$$

for some z. From axiom (19) in conjunction with the precondition axiom for $Stops(g)$, viz. (7), and the fact that we have just two sandglasses, axiom (1), which both were not running in S_0, axiom (4) in conjunction with (F4) and (5), it follows that

$$\begin{aligned} &ExpectedNatActions(c, t, Z_2) \equiv \\ &\quad c = Stops(G_{11}) \wedge t = 11 \vee c = Stops(G_7) \wedge t = 14 \end{aligned}$$

Given this notion, the macro $NextNatActions(c, t, z)$ stands for concurrent action c being all natural actions that happen in z at time t with t being the earliest timepoint at which natural actions are expected:

$$\begin{aligned} NextNatActions(c, t, z) \stackrel{\text{def}}{=}\ & ExpectedNatActions(c, t, z) \wedge \\ & \neg(\exists t', c')\,(ExpectedNatActions(c', t', z) \wedge t' < t\,) \end{aligned}$$

For example, $NextNatActions(Stops(G_{11}), 11, Z_2)$ according to what has been concluded above. On this basis, the collection of all natural actions that happen next determine one step further on a trajectory:

$$\begin{array}{l} Trajectory(z, z) \wedge \\ [\, Trajectory(z, z') \wedge NextNatActions(c, t, z') \supset \\ \qquad\qquad Trajectory(z, Succ(c, t, z'))\,] \end{array} \tag{F13}$$

The trajectory rooted in Z_2, e.g., thus contains $Succ(Stops(G_{11}), 11, Z_2)$—followed by the state $Succ(Stops(G_7), 14, Succ(Stops(G_{11}), 11, Z_2))$.

Axiom (F13) leaves open the possibility that arbitrary other states occur on a trajectory, too. The foundational axiom by which trajectories are linearized defines the function $ActualState$ of type $\textsc{sit} \times \textsc{real} \mapsto \textsc{state}$, which maps a situation s and a timepoint t to the actual state of the world in s at time t:

$$\begin{array}{l} Trajectory(State(s), z) \wedge t \geq Start(z) \\ \wedge\, (\forall a, t')\, (Natural(a) \wedge Poss(a, t, z) \supset t' > t) \\ \qquad\qquad \supset\ ActualState(s, t) = z \end{array} \tag{F14}$$

That is to say, the actual state z in situation s at time t must lie on the trajectory for s and must not have started later than t, and all natural actions expected in z must happen later. For example, from the above considerations it follows that if for some situation, say S_2, we have $State(S_2) = Z_2$, then $ActualState(S_2, 13) = Succ(Stops(G_{11}), 11, Z_2)$ and $ActualState(S_2, 15) = Succ(Stops(G_7), 14, Succ(Stops(G_{11}), 11, Z_2))$.

A complication raised by combining concurrency and natural actions is that natural actions may coincidentally happen at the very same time at which a (compound) deliberative action shall be performed. This needs to be taken into account both when verifying the preconditions of a deliberative action and when inferring its effect. For defining the preconditions, we introduce the expression $Poss(c, t, s)$, representing that a concurrent action c is possible in a situation s at time t.[10] This expression is a mere macro and is defined as follows:

$$\begin{array}{rl} Poss(c, t, s) \stackrel{\text{def}}{=} & (\forall a)(In(a, c) \supset \neg Natural(a)) \wedge \\ & (\forall c')\, (\, ExpectedNatActions(c', t, ActualState(s, t)) \supset \\ & \qquad\qquad Poss(c \cdot c', t, ActualState(s, t))\,) \end{array}$$

[10] Notice that thus far we have used the predicate $Poss$ with the third argument being a state.

That is, a collection of deliberative actions is possible just in case it is possible together with all natural actions expected at the very same time in the actual state. Accordingly, the state in the successor situation $Do(c,t,s)$ of performing concurrent deliberative action c at time t in situation s is obtained by adding to c all concurrent natural actions happening in the actual state of the world at time t:

$$\begin{aligned} Poss(c,t,s) \wedge ExpectedNatActions(c',t,ActualState(s,t)) \supset \\ State(Do(c,t,s)) = Succ(c \cdot c',t,ActualState(s,t)) \end{aligned} \tag{F15}$$

This completes the general theory of axiomatizing, by means of the Fluent Calculus, domains which involve continuous change and in which actions may happen concurrently. We are now in a position to extend the definition of planning by deduction in continuous worlds from [13] to continuous and concurrent domains: Consider a set of formulas $Axioms$ containing a domain specification and in particular given knowledge of the initial situation S_0. Furthermore, let $G(s)$ be a formula stipulating that the planning goal is achieved in situation s. Then a planning problem is defined as the problem of finding a term $\sigma = Do(c_n,t_n,\ldots,Do(c_1,t_1,S_0)\ldots)$ $(n \geq 0)$ such that[11]

$$Axioms \cup \{(\text{F1})\text{–}(\text{F15})\} \models POSS(\sigma) \wedge G(\sigma)$$

Let, for example, $Axioms$ be the domain axioms (1)–(19), and consider our original planning goal

$$G(s) \equiv (\exists t)\, Holds(Boiled(15),s,t)$$

where $Holds(f,s,t) \stackrel{\text{def}}{=} Holds(f,ActualState(s,t))$. The following situation can be inferred as a solution to this problem:

$$\begin{aligned} Do(EndBoiling,15,Do(Turn(G_7),11,Do(Turn(G_7),7, \\ Do(Turn(G_7) \cdot Turn(G_{11}) \cdot StartBoiling,0,S_0)))) \end{aligned}$$

Figure 3 depicts the sequence of state terms that occur during the execution of this solution.[12]

[11] Below, $POSS(\sigma) \stackrel{\text{def}}{=} Poss(c_1,t_1,S_0) \wedge \ldots \wedge Poss(c_n,t_n,Do(c_{n-1},t_{n-1},\ldots,S_0)\ldots)$.

[12] Here is a solution to the second planning problem, mentioned at the end of the introduction, of measuring 9 minutes with two sandglasses of a capacity of 4 and 7 minutes, respectively: $Do(EndBoiling,9,Do(Turn(G_7),8,Do(Turn(G_7),7,Do(Turn(G_4),4,$ $Do(Turn(G_4) \cdot Turn(G_7) \cdot StartBoiling,0,S_0)))))$.

$$Idle(G_7) \circ Idle(G_{11}) \circ StartTime(0) \circ z$$
$$\downarrow$$
$$Running(G_7, 7, 0) \circ Running(G_{11}, 11, 0) \circ Boiling(0) \circ StartTime(0) \circ z$$
$$\downarrow$$
$$Running(G_7, 7, 7) \circ Running(G_{11}, 11, 0) \circ Boiling(0) \circ StartTime(7) \circ z$$
$$\downarrow$$
$$Running(G_7, 4, 11) \circ Idle(G_{11}) \circ Boiling(0) \circ StartTime(11) \circ z$$
$$\downarrow$$
$$Idle(G_7) \circ Idle(G_{11}) \circ Boiled(15) \circ StartTime(15) \circ z$$

Figure 3: The sequence of states initially associated with the situations that occur in the course of our solution to the sandglass planning problem.

0.5 Summary

The Fluent Calculus has been proved versatile by extending its basic solution to the Frame Problem into various directions, such as nondeterminism and uncertainty, ramifications, knowledge and sensing, and continuous change. However, the existence of models for each one of these aspects does not imply that there be a unique model which covers them all. Rather the extensions have mostly been investigated in isolation. As a consequence, combining co-existing models for different phenomena can be a problem as challenging as addressing new aspects.

In this paper, we have introduced the concept of recursive state update axioms and reconciled it with the notion of process fluents and trajectories. The resulting Fluent Calculus allows for specifying and reasoning about domains involving continuous change and where actions occur concurrently. It remains an important challenge for future work to integrate the other existing Fluent Calculus extensions into a uniform, expressive axiomatization language for Cognitive Robotics.

Bibliography

[1] Andrew B. Baker. A simple solution to the Yale Shooting problem. In R. Brachman, H. J. Levesque, and R. Reiter, editors, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 11–20, Toronto, Kanada, 1989. Morgan Kaufmann.

[2] Wolfgang Bibel. Let's plan it deductively! *Artificial Intelligence*, 103(1–2):183–208, 1998.

[3] Volker Diekert and Grzegorz Rozenberg, editors. *The Book of Traces*. World Scientific, Singapore, 1995.

[4] Cordell Green. Theorem proving by resolution as a basis for question-answering systems. *Machine Intelligence*, 4:183–205, 1969.

[5] Steffen Hölldobler and Josef Schneeberger. A new deductive approach to planning. *New Generation Computing*, 8:225–244, 1990.

[6] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[7] Ray Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation*, pages 359–380. Academic Press, 1991.

[8] Ray Reiter. Natural actions, concurrency and continuous time in the situation calculus. In L. C. Aiello, J. Doyle, and S. Shapiro, editors, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 2–13, Cambridge, MA, November 1996. Morgan Kaufmann.

[9] Erik Sandewall. Combining logic and differential equations for describing real-world systems. In R. Brachman, H. J. Levesque, and R. Reiter, editors, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 412–420, Toronto, Kanada, 1989. Morgan Kaufmann.

[10] Murray Shanahan. A circumscriptive calculus of events. *Artificial Intelligence*, 77:249–284, 1995.

[11] Hans-Peter Störr and Michael Thielscher. A new equational foundation for the fluent calculus. In J. Lloyd etal, editor, *Proceedings of the International Conference on Computational Logic (CL)*, volume 1861 of *LNAI*, London (UK), July 2000. Springer.

[12] Michael Thielscher. Ramification and causality. *Artificial Intelligence*, 89(1–2):317–364, 1997.

[13] Michael Thielscher. Fluent Calculus planning with continuous change. *Electronic Transactions on Artificial Intelligence*, 1999. (Submitted.) URL: `http://www.ep.liu.se/ea/cis/1999/011/`.

[14] Michael Thielscher. From Situation Calculus to Fluent Calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence*, 111(1–2):277–299, 1999.

[15] Michael Thielscher. Nondeterministic actions in the fluent calculus: Disjunctive state update axioms. In S. Hölldobler, editor, *Intellectics and Computational Logic*, pages 327–345. Kluwer Academic, 2000.

[16] Michael Thielscher. Representing the knowledge of a robot. In A. Cohn, F. Giunchiglia, and B. Selman, editors, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 109–120, Breckenridge, CO, April 2000. Morgan Kaufmann.