A Formal Assessment Result for Fluent Calculus Using the Action Description Language $\mathcal{A}_k$ [⋆]

Ozan Kahramanoğulları and Michael Thielscher

Department of Computer Science
Dresden University of Technology

Abstract. Systematic approaches like the family of *Action Description Languages* have been designed for the formal assessments of action calculi. We assess the fluent calculus for knowledge and sensing with the help of the recently developed, high-level action language $\mathcal{A}_k$. As the main result, we present a provably correct embedding of this language into fluent calculus, excluding the while loops in the query language. As a spin-off, the action programming language FLUX, which is based on fluent calculus, provides a system for answering queries to $\mathcal{A}_k$ domains. Conversely, the action description language may serve as a high-level surface language for specifying action domains in FLUX.

1 Introduction

An unsatisfactory aspect of research into reasoning about actions is the co-existence of a variety of different approaches, which are difficult to assess and compare. Systematic approaches such as [11] or the Action Description Language $\mathcal{A}$ [2] have been developed to help eliminate this deficiency. Providing a high-level but formal semantics, these approaches are intended to be used to prove correctness of different action calculi for well-defined problem classes. A formal evaluation is particularly important when it comes to modeling complex phenomena such as knowledge and sensing actions.

In this paper, we follow this systematic approach and assess the fluent calculus for knowledge and sensing [12]. To this end, we use the extended Action Description Language $\mathcal{A}_k$ developed in [9] as a minimal extension of the action description language $\mathcal{A}$ [2] to handle non-deterministic actions, knowledge, and sensing. We present a mapping from $\mathcal{A}_k$ domains and queries (without loops) into fluent calculus and, as the main result, prove soundness and completeness wrt. the high-level semantics. Doing so, we show that fluent calculus can express an intricate commonsense phenomena of sensing captured by $\mathcal{A}_k$, namely, the unknown preconditions of a sensing action being learned during sensing. In addition, we have developed a logic programming realization of this translation into the action programming language FLUX—the <u>Flu</u>ent E<u>x</u>ecutor [13]. Our achievement is three-fold:

[⋆] A preliminary version of this paper has been presented at the 2003 AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning.

1. The result shows that fluent calculus for knowledge and sensing is correct wrt. the problem class defined by $\mathcal{A}_k$.
2. Augmented by the translation program, FLUX provides a system for answering queries to $\mathcal{A}_k$ domains.
3. The syntax of $\mathcal{A}_k$ can serve as a high-level surface language for specifying action domains in FLUX.

The rest of the paper is organized as follows. We begin by recapitulating basic notions and notations of $\mathcal{A}_k$ and fluent calculus, respectively. We then present a provably correct translation of domain descriptions in $\mathcal{A}_k$ into fluent calculus axiomatizations. Thereafter, we extend the translation to queries in $\mathcal{A}_k$ so that query entailment in fluent calculus is sound and complete wrt. the high-level semantics. We conclude with a sketch of the logic programming realization of our translation using FLUX, followed by a brief discussion and outlook.

2 The Action Description Language $\mathcal{A}_k$

The language $\mathcal{A}_k$ [9] is the minimal extension of the action description language $\mathcal{A}$ [2] to handle non-deterministic actions, knowledge, and sensing. Domain descriptions in $\mathcal{A}_k$ allow one to answer queries about what will hold after executing a sequence of actions of that domain description.

2.1 Syntax of the Domain Language

The language of $\mathcal{A}_k$ consists of two non-empty disjoint sets of symbols **F**, **A**. They are called *fluents* and *actions*, respectively. The set **A** consists of two disjoint sets of actions: *Sensing* actions and *non-sensing* actions. Actions will be generically denoted by a, possibly indexed. A *fluent literal* is an element from the set of fluents that is possibly preceded by a $\neg$ sign. Fluents will be denoted by f or g and fluent literals by ℓ, p, or q, all possibly indexed.

There are three kinds of propositions in $\mathcal{A}_k$: A *value proposition* is an expression of the form

$$\textbf{initially } \ell \tag{1}$$

where ℓ denotes a fluent literal. Value propositions describe the initial knowledge the agent has about the world.

Effect propositions are expressions of the form

$$a \textbf{ causes } \ell \textbf{ if } p_1, \ldots, p_n \tag{2}$$

$$a \textbf{ may affect } f \textbf{ if } p_1, \ldots, p_n \tag{3}$$

where a is a non-sensing action, ℓ and $p_1, \ldots, p_n$ $(n \geq 0)$ are fluent literals, and f is a fluent. Intuitively, the first expression above mean that in a state where $p_1, \ldots, p_n$ are true, the execution of a causes ℓ to become true. The second expression says that the truth value of f may indeterminately change if a is executed in a state where $p_1, \ldots, p_n$ are true.

Sensing actions are described by *knowledge laws*:

$$a_s \textbf{ causes to know } f \textbf{ if } p_1, \ldots, p_n \tag{4}$$

where a_s is a sensing action, f is a fluent and $p_1, \ldots, p_n$ are preconditions as in (2) and (3). This expression says that if $p_1, \ldots, p_n$ are true, then the execution of a_s causes the agent to realize the current value of f in the world. Sensing actions are not allowed to occur in any effect proposition. A collection of the above propositions and laws is called a *domain description*.

Example Consider a robot that faces the door of the room but does not know if the door is open or not. The robot can execute the sensing action *Look*, which makes it realize the door's being open or closed, provided it is facing the door. The robot can also execute the action *SendId*, which is the action of sending the electronic signal of the door. This action causes the door to open if it is closed, and to close if it is open:

$$D_1 = \begin{cases} r_1: \textbf{ initially } \mathit{FacingDr} \\ r_2: \mathit{SendId} \textbf{ causes } \mathit{DrOpn} \textbf{ if } \neg \mathit{DrOpn} \\ r_3: \mathit{SendId} \textbf{ causes } \neg \mathit{DrOpn} \textbf{ if } \mathit{DrOpn} \\ r_4: \mathit{Look} \textbf{ causes to know } \mathit{DrOpn} \textbf{ if } \mathit{FacingDr} \end{cases}$$

2.2 Semantics of the Domain Language

The semantics of $\mathcal{A}_k$ explains how an agent's knowledge changes according to the effects of the actions defined by a domain description. A *state* is a set of fluents. A set of states represents a *situation*. This way, incomplete descriptions of the world are captured.[2] The knowledge of the robot is represented by a set of (possibly incomplete) worlds (i.e., situations) in which the agent believes that it can be. Such a set is called an *epistemic state*. For example, the epistemic state (a) in Figure 1 represents that the robot knows *FacingDr* but is ignorant of whether or not *DrOpn* holds. In contrast, in the epistemic state (b) the robot knows *whether* the door is open: In situation Σ_1 fluent *DrOpn* is false in all states. Conversely, in situation Σ_2 fluent *DrOpn* is true in all states. Hence, in all situations of epistemic state (b) the status of *DrOpn* is known.

A fluent f holds in a state σ iff $f \in \sigma$ (denoted by: $\sigma \models_{\mathcal{A}_k} f$). A fluent f does not hold in a state σ iff $f \notin \sigma$ (denoted by: $\sigma \not\models_{\mathcal{A}_k} f$). The truth of a formula φ made of fluents and the standard logical connectives in a state is recursively defined as usual. A formula φ is true in a situation Σ (denoted by: $\Sigma \models_{\mathcal{A}_k} \varphi$) if the formula is true in every state in Σ; it is false if $\neg\varphi$ is true in every state in Σ. A formula is true in an epistemic state if it is true in every situation in the epistemic state; it is false if its negation is true. A situation is *consistent* if it is non-empty, otherwise it is *inconsistent*. A situation is *complete* if it contains only one state, otherwise it is *incomplete*.

[2] Unfortunately, the inventors of $\mathcal{A}_k$ decided to use the term *situation* to denote something very different from what is normally called a situation in action calculi, namely, a sequence of actions. The reader should not be confused by this clash of names.

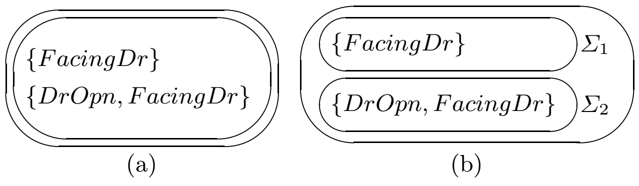

Fig. 1. Two epistemic states.

Interpretations for $\mathcal{A}_k$ are transition functions that map pairs of actions and situations into situations. In order to interpret the effects of the actions at state level, *0-interpretations* map actions a and states σ into sets of states $\Phi_0(a, \sigma)$. Each state in $\Phi_0(a, \sigma)$ represents a possible result of performing a in σ; this is how nondeterministic actions are dealt with.

Definition 1 *A 0-interpretation Φ_0 is a* 0-model *of a domain description D iff for every state σ and action a, $\Phi_0(a, \sigma)$ is the set that contains every state σ' such that:*

1. *For a fluent f of any effect proposition of the form 'a* **causes** *f* **if** *$p_1, \ldots, p_n$' in D, the fluent f holds in σ' if the preconditions $p_1, \ldots, p_n$ hold in σ;*
2. *For a fluent literal $\neg f$ of any effect proposition of the form 'a* **causes** *$\neg f$* **if** *$p_1, \ldots, p_n$' in D, the fluent f does not hold in σ' if the preconditions $p_1, \ldots, p_n$ hold in σ;*
3. *For a fluent f such that there are no effect propositions of the above types, $f \in \sigma'$ iff $f \in \sigma$ unless there is a non-deterministic effect proposition of the form 'a* **may affect** *f* **if** *$p_1, \ldots, p_n$' for which $p_1, \ldots, p_n$ hold in σ.*

According to item 3, if there is no applicable deterministic effect proposition, then f keeps its truth-value unless there is an applicable non-deterministic effect proposition concerning f, in which case f can take on either truth-value in σ', thus giving rise to several possible resulting states.

Knowledge laws are interpreted according to the following definitions. An important aspect of the definition of sensing in $\mathcal{A}_k$ is that the *conditions* under which a fluent can be sensed will become known to the agent if the value of the fluent being sensed is previously unknown. For example, if our robot truly learns whether the door is open by performing the sensing action *Look*, then it also realizes that it is facing the door.

Definition 2 *Let Σ be a consistent situation, f a fluent, and φ a disjunction of conjunctions of fluent literals (preconditions). A consistent situation Σ' is 'f, φ – compatible' with Σ iff the following holds.*

- *If f is either true or false in Σ then $\Sigma' = \Sigma$.*
- *If f is unknown (neither true nor false) in Σ then Σ' must satisfy one of the following conditions:*

1. $\Sigma' = \{\sigma \in \Sigma \mid \varphi \text{ is not true in } \sigma\}$
2. $\Sigma' = \{\sigma \in \Sigma \mid \varphi \text{ is true in } \sigma,\ f \notin \sigma\}$
3. $\Sigma' = \{\sigma \in \Sigma \mid \varphi \text{ is true in } \sigma,\ f \in \sigma\}$

In other words, if the fluent being sensed is unknown, then the sensing action splits situation Σ into three situations: One in which φ is false, one in which φ is true and f is false, and one in which φ is true and f is true. These situations are the worlds that are considered possible after the execution of the sensing action. Then the set containing all these situations is the epistemic state.

Definition 3 *A fluent f is a* potential sensing effect *of a sensing action a_s in a domain D if there is a knowledge law of the form a_s* **causes to know** *f* **if** *φ in D. The knowledge precondition of a fluent f with respect to a sensing action a_s in a domain D is $\varphi_1 \vee \ldots \vee \varphi_n$ where*

$$\{a_s \textbf{ causes to know } f \textbf{ if } \varphi_1, \ldots, a_s \textbf{ causes to know } f \textbf{ if } \varphi_n\}$$

are the knowledge laws with action a_s and potential sensing effect f.

Definition 4 *Given an interpretation Φ of $\mathcal{A}_k$, Φ is a* model of domain description *D, if and only if it satisfies the following: $\Phi(a, \Sigma) = \emptyset$ if Σ inconsistent, and for any consistent situation Σ:*

1. *There exists a 0-model Φ_0 of D, such that for any non-sensing action a, $\Phi(a, \Sigma) = \bigcup_{\sigma \in \Sigma} \Phi_0(a, \sigma)$.*
2. *For each sensing action a_s, let $f_1, \ldots, f_n$ be the potential sensing effects of a_s and φ_i the knowledge precondition of f_i with respect to a_s. Then $\Phi(a_s, \Sigma)$ must be consistent, and if $n = 0$ then $\Phi(a_s, \Sigma) = \Sigma$ otherwise $\Phi(a_s, \Sigma) = \bigcap_{i \in [1..n]} \Sigma_i$, such that each Σ_i is a situation $f_i, \varphi_i -$ compatible with Σ.*

Definition 5 *A state is called* initial state *of a domain description D iff for every value proposition of the form '***initially** *ℓ' in D, ℓ is true in σ. The initial situation Σ_0 of D is the set of all the initial states of D.*

2.3 The Query Language

Queries in $\mathcal{A}_k$ are of the form φ **after** α where φ is a conjunction of fluent literals and α is a plan, which is inductively defined as follows: The empty sequence $[\,]$ is a plan. If a is an action and α is a plan then the concatenation $[a|\alpha]$ is a plan. If φ is a conjunction of fluent literals and α, α_1, and α_2 are plans then [**if** φ **then** $\alpha_1 \mid \alpha$] and [**if** φ **then** α_1 **else** $\alpha_2 \mid \alpha$] are (conditional) plans.[3]

Definition 6 *The plan evaluation function Γ_Φ of an interpretation Φ is a function such that for any situation Σ:*

1. *$\Gamma_\Phi([\,], \Sigma) = \Sigma$.*

[3] The full query language of $\mathcal{A}_k$ includes plans with loops, which we do not consider in this paper.

2. $\mathit{\Gamma}_{\mathit{\Phi}}([a|\alpha],\mathit{\Sigma}) = \mathit{\Gamma}_{\mathit{\Phi}}(\alpha,\mathit{\Phi}(a,\mathit{\Sigma}))$ *for any action* a.

3. $\mathit{\Gamma}_{\mathit{\Phi}}([\ \textbf{if}\ \varphi\ \ \textbf{then}\ \alpha_1 \mid \alpha],\mathit{\Sigma}) = \mathit{\Gamma}_{\mathit{\Phi}}(\alpha,\mathit{\Sigma}')$, *where*

$$\mathit{\Sigma}' = \begin{cases} \mathit{\Gamma}_{\mathit{\Phi}}(\alpha_1,\mathit{\Sigma}) & \textit{if } \varphi \textit{ is true in } \mathit{\Sigma} \\ \mathit{\Sigma} & \textit{if } \varphi \textit{ is false in } \mathit{\Sigma} \\ \emptyset & \textit{otherwise} \end{cases}$$

4. $\mathit{\Gamma}_{\mathit{\Phi}}([\ \textbf{if}\ \varphi\ \ \textbf{then}\ \alpha_1\ \textbf{else}\ \alpha_2 \mid \alpha],\mathit{\Sigma}) = \mathit{\Gamma}_{\mathit{\Phi}}(\alpha,\mathit{\Sigma}')$, *where*

$$\mathit{\Sigma}' = \begin{cases} \mathit{\Gamma}_{\mathit{\Phi}}(\alpha_1,\mathit{\Sigma}) & \textit{if } \varphi \textit{ is true in } \mathit{\Sigma} \\ \mathit{\Gamma}_{\mathit{\Phi}}(\alpha_2,\mathit{\Sigma}) & \textit{if } \varphi \textit{ is false in } \mathit{\Sigma} \\ \emptyset & \textit{otherwise} \end{cases}$$

Definition 7 *A query '*φ **after** α*' is entailed by a domain description* D *(written as:* $D \models_{\mathcal{A}_k} \varphi$ **after** α*) iff for every model* $\mathit{\Phi}$ *of* D*,* φ *is true in* $\mathit{\Gamma}_{\mathit{\Phi}}(\alpha,\mathit{\Sigma}_0)$ *where* $\mathit{\Sigma}_0$ *is the initial situation of* D*.*

Example (continued) As shown in detail in [9], D_1 entails[4]

$$\mathit{DrOpn}\ \textbf{after}\ [\mathit{Look}, \textbf{if}\ \neg \mathit{DrOpn}\ \textbf{then}\ [\mathit{SendId}]] \tag{5}$$

3 Fluent Calculus

Fluents and states. A many-sorted predicate logic language, fluent calculus extends the classical situation calculus by the concept of a state. Its signature includes the standard sorts FLUENT and STATE. The intuition is that a state is characterized by the fluents that hold in it. Formally, every term of sort FLUENT also belongs to sort STATE, and the signature contains the two standard functions $\emptyset$: STATE (denoting the empty state) and $\circ$: STATE $\times$ STATE $\mapsto$ STATE (written in infix notation and denoting the union of two states). Let $\mathit{Holds}(f, z)$ be an abbreviation for the equational formula $(\exists z')\, z = f \circ z'$, then the *foundational axioms* of fluent calculus are:[5]

$$\begin{array}{ll} (z_1 \circ z_2) \circ z_3 = z_1 \circ (z_2 \circ z_3) & z_1 \circ z_2 = z_2 \circ z_1 \\ \neg \mathit{Holds}(f, \emptyset) & \mathit{Holds}(f, g) \supset f = g \\ \mathit{Holds}(f, z_1 \circ z_2) \ \supset\ \mathit{Holds}(f, z_1) \vee \mathit{Holds}(f, z_2) & \\ (\forall f)\,(\mathit{Holds}(f, z_1) \equiv \mathit{Holds}(f, z_2)) \ \supset\ z_1 = z_2 & \\ (\forall P)(\exists z)(\forall f)\,(\mathit{Holds}(f, z) \equiv P(f)) & \end{array}$$

The very last axiom, with P being a second-order predicate variable of sort FLUENT, stipulates the existence of a state for any set of fluents.

Actions and situations. The sorts ACTION and SIT (for situations) are inherited from situation calculus along with the standard functions S_0 : SIT (denoting

[4] For the sake of readability, we use the standard Prolog list syntax where $[a_1, \ldots, a_n]$ is $[a_1| \ldots [a_n|[\,]]]$.

[5] Below, f, g are FLUENT variables while z_1, z_2, z_3 are STATE variables. Free variables are universally quantified.

the initial situation) and $Do : \textsc{action} \times \textsc{sit} \mapsto \textsc{sit}$ (denoting the successor situation of performing an action). In addition, fluent calculus includes the special function $State : \textsc{sit} \mapsto \textsc{state}$ to denote the state of the world in a situation. With this, the basic predicate $Holds(f, s)$ of situation calculus is modeled as a mere macro in fluent calculus which stands for $Holds(f, State(s))$. For example, the statement $Holds(FacingDr, S_0)$ means

$$(\exists z)\, State(S_0) = FacingDr \circ z \tag{6}$$

State update axioms. Fluent calculus provides a solution to the fundamental frame problem in classical logic. The key is a purely axiomatic characterization of removal and addition of fluents to states. Let z_1, z_2 be states and f a fluent, then the expression $z_1 - f = z_2$ (denoting removal of f from z_1) is defined as an abbreviation for the formula

$$[z_2 = z_1 \vee z_2 \circ f = z_1] \wedge \neg Holds(f, z_2)$$

Let $\vartheta^- = f_1 \circ \ldots \circ f_n$ $(n \geq 0)$, then an inductive extension of this macro defines $z_1 - (\vartheta^- \circ f) = z_2$ as $(\exists z)\, (z_1 - \vartheta^- = z \wedge z - f = z_2)$. Finally, the definition of the *update equation* $z_2 = z_1 - \vartheta^- + \vartheta^+$, where $\vartheta^+ = g_1 \circ \ldots \circ g_m$ $(m \geq 0)$, is given by $(\exists z)\, (z_1 - \vartheta^- = z \wedge z_2 = z \circ \vartheta^+)$. The frame problem is then solved by *state update axioms*, which use update equations to specify the effects of an action. For example, the following state update axiom specifies the (conditional) effect of $SendId$ for all situations s:[6]

$$\begin{array}{rl} & \neg Holds(DrOpn, s) \wedge State(Do(SendId, s)) = State(s) + DrOpn \\ \vee & Holds(DrOpn, s) \wedge State(Do(SendId, s)) = State(s) - DrOpn \end{array} \tag{7}$$

Representing knowledge. Basic fluent calculus has been extended in [12] by the foundational predicate $KState : \textsc{sit} \times \textsc{state}$ to allow for both representing state knowledge and reasoning about actions which involve sensing. An instance $KState(s, z)$ means that, according to the knowledge of the agent, z is a *possible* state in situation s. For example, the initial knowledge of our robot can be specified by this axiom:

$$(\forall z)\, (KState(S_0, z) \equiv Holds(FacingDr, z)) \tag{8}$$

That is to say, all states which satisfy the initial specification are to be considered possible by the robot. In particular, it is unknown whether or not the door in question is open. Generally, a fluent is known to hold in a situation (not to hold, respectively) just in case it is true (false, respectively) in all possible states:

$$\begin{array}{rcl} Knows(f, s) & \overset{\text{def}}{=} & (\forall z)\, (KState(s, z) \supset Holds(f, z)) \\ Knows(\neg f, s) & \overset{\text{def}}{=} & (\forall z)\, (KState(s, z) \supset \neg Holds(f, z)) \\ Unknown(f, s) & \overset{\text{def}}{=} & \neg Knows(f, s) \wedge \neg Knows(\neg f, s) \end{array}$$

[6] For the sake of simplicity, we ignore the concept of action preconditions in this paper, since actions in $\mathcal{A}_k$ are always executable. Empty positive and negative effects of actions in update equations are simply omitted.

The frame problem for knowledge is solved by axioms that determine the relation between the possible states before and after an action, be it sensing or not. An example of such a *knowledge update axiom* is

$$\begin{array}{l} \mathit{KState}(\mathit{Do}(\mathit{Look}, s), z) \equiv \\ \quad (\mathit{KState}(s, z) \wedge \\ \quad\; [\mathit{Holds}(\mathit{FacingDr}, s) \supset (\mathit{Holds}(\mathit{DrOpn}, z) \equiv \mathit{Holds}(\mathit{DrOpn}, s))] \wedge \\ \quad\; [\mathit{Unknown}(\mathit{DrOpn}, s) \supset (\mathit{Holds}(\mathit{FacingDr}, z) \equiv \mathit{Holds}(\mathit{FacingDr}, s))]\,) \end{array} \tag{9}$$

Put in words, if the robot faces the door, then all possible states in the resulting situation agree with the actual situation on whether *DrOpn* holds. Moreover, if it is unknown whether the door is open, then a side-effect of the sensing action is that the robot learns the condition of the conditional effect, that is, whether or not *FacingDr* holds. A pure sensing action, *Look* strictly reduces the set of possible states according to axiom (9). In general, however, knowledge update axioms may define arbitrary changes in the set of possible states, so that knowledge may also get lost after performing the action, e.g., in case of non-deterministic actions.

4 From $\mathcal{A}_k$ Domains to Fluent Calculus

In this section, we present a translation function which maps an $\mathcal{A}_k$ domain description D into a set of fluent calculus axioms $\Pi(D)$. To begin with, the fluents $\{f_1, \ldots, f_n\}$ and actions $\{a_1, \ldots, a_m\}$ in D determine the *unique-name axioms* $\Pi_{una}(D) = \{\bigwedge_{i \neq j} f_i \neq f_j \;,\; \bigwedge_{i \neq j} a_i \neq a_j\}$.

4.1 Translating the Value Propositions

The axioms for the initial state and the initial knowledge state, respectively, are determined by the value propositions: Let

$$\left\{ \begin{array}{l} \textbf{initially } f_1, \ldots, \textbf{initially } f_n, \\ \textbf{initially } \neg g_1, \ldots, \textbf{initially } \neg g_m \end{array} \right\}$$

be the set of all the value propositions of D, then the set $\Pi_{init}(D)$ contains these two axioms:

$$\begin{array}{l} (\exists z)\,(\mathit{State}(S_0) = f_1 \circ \ldots \circ f_n \circ z \wedge \\ \qquad\qquad \neg \mathit{Holds}(g_1, z) \wedge \ldots \wedge \neg \mathit{Holds}(g_m, z)) \\ \mathit{KState}(S_0, z) \equiv \mathit{Holds}(f_1, z) \wedge \ldots \wedge \mathit{Holds}(f_n, z) \\ \qquad\qquad \wedge \neg \mathit{Holds}(g_1, z) \wedge \ldots \wedge \neg \mathit{Holds}(g_m, z) \end{array}$$

Example (continued) The domain description D_1 contains only one value proposition, which determines the set $\Pi_{init}(D_1)$ as the fluent calculus axioms (6) and (8) mentioned earlier.

4.2 Translating the Effect Propositions

The translation handles each action a of an $\mathcal{A}_k$ domain description separately. Since sensing actions are not allowed to occur in any effect proposition, effect propositions are translated independently from the knowledge laws. As the first step for translating the effect propositions, we define a set which summarizes the effects of a non-sensing action of an $\mathcal{A}_k$ domain description:

Definition 8 *Let D be an $\mathcal{A}_k$ domain description. Let*

$$\left\{\begin{array}{l} a\ \textbf{causes}\ \ell_1\ \textbf{if}\ \varphi_1,\ \ldots,\ a\ \textbf{causes}\ \ell_m\ \textbf{if}\ \varphi_m, \\ a\ \textbf{may affect}\ f_1\ \textbf{if}\ \phi_1,\ \ldots,\ a\ \textbf{may affect}\ f_n\ \textbf{if}\ \phi_n \end{array}\right\}$$

be the set of all effect propositions for the action a in D, where each ℓ_i is a fluent literal, each f_j is a fluent, each φ_i and ϕ_j is a sequence of fluent literals $(1 \le i \le m, 1 \le j \le n)$. The effect bag *of action a (denoted by $\mathcal{B}_a$) is the set*

$$\begin{array}{l} \mathcal{B}_a = \{(\ell_1, S_1, dt), \ldots, (\ell_n, S_m, dt), \\ \qquad\qquad (f_1, S_{m+1}, nd), \ldots, (f_m, S_{n+m}, nd)\}, \end{array}$$

where $S_i = \{p_1, \ldots, p_k\}$ for each sequence of fluent literals φ_i, and ϕ_j of the form $p_1, \ldots, p_k$. The tags 'dt' and 'nd' signify the deterministic effect and the non-deterministic effect, respectively.

The next step is to consider all combinations of preconditions of a. Given the effect bag $\mathcal{B}_a = \{(\ell_1, S_1, dt), \ldots, (\ell_n, S_n, nd)\}$, let Θ_a be all fluent literals that occur in some precondition S_i $(1 \le i \le n)$. Let $\mathcal{P}_a = \{P_1, \ldots, P_n\}$ be the power-set of Θ_a, and for each P_i let $\Theta_a \setminus P_i = \{f_{i,1}, \ldots, f_{i,k}\}$. Then the *condition set* of an action a of an $\mathcal{A}_k$ domain description is the set $\mathcal{C}_a = \{C_1, \ldots, C_n\}$, where for each $i = 1 \ldots n$

$$C_i = P_i \cup \{\neg f_{i,1}, \ldots, \neg f_{i,k}\}$$

The condition set $\mathcal{C}_a$ contains all the combinations of fluent literals under which the execution of the action a has an effect. Then, for each element C_i of $\mathcal{C}_a$, the sets $\mathcal{F}_i^{dt}$ and $\mathcal{F}_i^{nd}$ shall contain the fluents that are affected or that may be affected, respectively, by the execution of action a under the condition C_i:

$$\begin{array}{l} \mathcal{F}_i^{dt} = \{f_k \mid (f_k, S_k, dt) \in \mathcal{B}_a, S_k \subseteq C_i\} \\ \mathcal{F}_i^{nd} = \{f_k \mid (f_k, S_k, nd) \in \mathcal{B}_a, S_k \subseteq C_i\} \end{array}$$

The set of fluent literals that represents the deterministic effects of the action a under the condition C_i, viz. $\mathcal{F}_i^{dt}$, can be partitioned into two sets of positive and negative effects. That is, if $\mathcal{F}_i^{dt} = \{f_1, \ldots, f_n, \neg g_1, \ldots, \neg g_m\}$, then

$$\mathcal{F}_i^{+} = \{f_1, \ldots, f_n\} \text{ and } \mathcal{F}_i^{-} = \{g_1, \ldots, g_m\}$$

Definition 9 *A* pre-update *Λ_a for an action a of an $\mathcal{A}_k$ domain description D is the set*

$$\Lambda_a = \{(C_1, \mathcal{F}_1^{+}, \mathcal{F}_1^{-}, \mathcal{F}_1^{nd}), \ldots, (C_n, \mathcal{F}_n^{+}, \mathcal{F}_n^{-}, \mathcal{F}_n^{nd})\}$$

A pre-update Λ_a of an action a contains all effects of the action a under the different conditions. However, the non-deterministic effects of actions have been treated separately so far. In the next step, we introduce the recursive function μ that allows branching for a non-deterministic effect of an action. That is, given that a fluent f is a non-deterministic effect of an action a under precondition C_i, if f does not occur in the positive effects $\mathcal{F}_n^+$ nor in the negative effects $\mathcal{F}_n^-$, then there will be two possible effects for this action under precondition C_i: One in which f becomes a positive effect and one in which f becomes a negative effect.

Definition 10 *Let $\mu(C_i, \mathcal{F}_i^+, \mathcal{F}_i^-, \mathcal{F}_i^{nd})$ be defined as*

$$\begin{cases} \{(C_i, \mathcal{F}_i^+, \mathcal{F}_i^-)\} \\ \qquad\qquad \textit{if} \quad \mathcal{F}_i^{nd} = \emptyset \\ \mu((C_i, \mathcal{F}_i^+, \mathcal{F}_i^-, \mathcal{F}_i^{nd} \setminus \{f\})) \\ \qquad\qquad \textit{if} \quad f \in \mathcal{F}_i^{nd}, \ \textit{and either } f \in \mathcal{F}_i^+ \textit{ or } f \in \mathcal{F}_i^-) \\ \mu((C_i, \mathcal{F}_i^+ \cup \{f\}, \mathcal{F}_i^-, \mathcal{F}_i^{nd} \setminus \{f\})) \\ \ \cup \mu((C_i, \mathcal{F}_i^+, \mathcal{F}_i^- \cup \{f\}, \mathcal{F}_i^{nd} \setminus \{f\})) \\ \qquad\qquad \textit{if} \quad f \in \mathcal{F}_i^{nd}\ , \ f \notin \mathcal{F}_i^+, \ \textit{and } f \notin \mathcal{F}_i^- \end{cases}$$

Then, given a pre-update $\Lambda_a = \{\gamma_1, \ldots, \gamma_n\}$, an update *for an action a is the set $\Omega_a = \bigcup_{i=1}^n \mu(\gamma_i)$.*

An update Ω_a of a non-sensing action a of an $\mathcal{A}_k$ domain description contains all the information to construct the fluent calculus state update and knowledge update axioms for this action. Before, we proceed with constructing these axioms from an update Ω_a, below we illustrate the translation up to this point on the domain description D_1.

Example (continued) The effect bag $\mathcal{B}_{SendId}$ is

$$\{(DrOpn, \{\neg DrOpn\}, dt), (\neg DrOpn, \{DrOpn\}, dt)\}$$

Then $\Theta_{SendId} = \{DrOpn\}$, which is the set of all the fluents that appear in the conditions of the effect propositions for action $SendId$; hence, $\mathcal{P}_{SendId} = \{\{\}, \{DrOpn\}\}$. From this we obtain the *condition set*

$$\mathcal{C}_{SendId} = \{\{\neg DrOpn\}, \{DrOpn\}\}$$

After getting the *pre-update*

$$\Lambda_{SendId} = \{(\{\neg DrOpn\}, \{DrOpn\}, \{\}, \{\}), \ (\{DrOpn\}, \{\}, \{DrOpn\}, \{\})\}$$

according to Definition 9, plugging all the elements of it into function μ, and taking their union, we obtain the *update* Ω_{SendId} according to Definition 10:

$$\begin{aligned} \Omega_{SendId} = \{(&\{\neg DrOpn\}, \{DrOpn\}, \{\}), \\ &(\{DrOpn\}, \{\}, \{DrOpn\})\}. \end{aligned} \tag{10}$$

From Ω_a to State and Knowledge Update Axioms In $\mathcal{A}_k$ it is not possible to differentiate between the actual effects of actions and what an agent knows of the effects of an action. Hence, in our translation state update axioms and knowledge update axioms coincide as far as non-sensing actions are concerned.

Let Ω_a be the update of a non-sensing action a of an $\mathcal{A}_k$ domain description D. The translation algorithm generates the state update axiom and the knowledge update axiom of action a from the update Ω_a as follows.

Let $\Omega_a = \{(C_1, \mathcal{F}_1^+, \mathcal{F}_1^-), \ldots, (C_n, \mathcal{F}_n^+, \mathcal{F}_n^-)\}$, where for each $i = 1 \ldots n$,

$$\begin{aligned} C_i &= \{p_{i,1}, \ldots, p_{i,m_i}\} \\ \mathcal{F}_i^+ &= \{f_{i,1}, \ldots, f_{i,k_i}\} \\ \mathcal{F}_i^- &= \{g_{i,1}, \ldots, g_{i,l_i}\} \end{aligned}$$

These are the resulting state and knowledge update axioms for action a in $\Pi_{update}(D)$:

$$\begin{array}{l} HOLDS(p_{1,1} \wedge \ldots \wedge p_{1,m_1}, s) \wedge \\ \quad State(Do(a,s)) = (State(s) - g_{1,1} \circ \ldots \circ g_{1,l_1}) \\ \qquad\qquad\qquad\qquad\qquad\qquad + f_{1,1} \circ \ldots \circ f_{1,k_1} \\ \vee \ \ldots \ \vee \\ HOLDS(p_{n,1} \wedge \ldots \wedge p_{n,m_n}, s) \wedge \\ \quad State(Do(a,s)) = (State(s) - g_{n,1} \circ \ldots \circ g_{n,l_n}) \\ \qquad\qquad\qquad\qquad\qquad\qquad + f_{n,1} \circ \ldots \circ f_{n,k_n} \end{array}$$

$$\begin{array}{l} KState(Do(a,s), z) \ \equiv \ (\exists z')(KState(s, z') \wedge \\ \quad [\, HOLDS(p_{1,1} \wedge \ldots \wedge p_{1,m_1}, z') \wedge \\ \qquad z = (z' - g_{1,1} \circ \ldots \circ g_{1,l_1}) + f_{1,1} \circ \ldots \circ f_{1,k_1} \\ \quad \vee \ \ldots \ \vee \\ \quad Holds(p_{n,1} \wedge \ldots \wedge p_{n,m_n}, z') \wedge \\ \qquad z = (z' - g_{n,1} \circ \ldots \circ g_{n,l_n}) + f_{n,1} \circ \ldots \circ f_{n,k_n} \,])) \end{array}$$

where $HOLDS(\varphi, s)$ and $HOLDS(\varphi, z)$, respectively, is obtained from φ by substituting each occurrence of a fluent f by $Holds(f, s)$ and $Holds(f, z)$, respectively.

Example (continued) Given the update Ω_{SendId} of equation (10), as the state update axiom for $SendId$ we obtain formula (7) mentioned earlier, while the knowledge update axiom is

$$\begin{array}{l} KState(Do(SendId, s), z) \ \equiv \ (\exists z')(KState(s, z') \wedge \\ \quad [\, \neg Holds(DrOpn, z') \wedge z = z' + DrOpn \ \vee \\ \quad\ Holds(DrOpn, z') \wedge z = z' - DrOpn \,]) \end{array}$$

4.3 Translating the Knowledge Laws

We begin the translation of the knowledge laws by defining a set which contains all the effects of a sensing action.

Definition 11 *Let D be an $\mathcal{A}_k$ domain description. Let*

$$\left\{\begin{array}{l} a \textbf{ causes to know} f_1 \textbf{ if } P_{1,1} \\ \quad\vdots \\ a \textbf{ causes to know} f_1 \textbf{ if } P_{1,n_1} \\ \qquad\qquad\vdots \\ a \textbf{ causes to know } f_m \textbf{ if } P_{m,1} \\ \quad\vdots \\ a \textbf{ causes to know } f_m \textbf{ if } P_{m,n_m} \end{array}\right\}$$

be the set of all the knowledge laws for the sensing action a in D, where each f_i is a fluent, and each $P_{i,j}$ is sequence of fluent literals $(1 \leq i \leq m; 1 \leq j \leq n_i)$. The knowledge bag *of sensing action a (denoted by $\mathcal{K}_a$) is the set*

$$\mathcal{K}_a = \{(f_1, \varphi_1), \ldots, (f_m, \varphi_m)\} \tag{11}$$

where for each $i = 1 \ldots m$, $\varphi_i = C_{i,1} \vee \ldots \vee C_{i,n_i}$ such that for each $j = 1 \ldots n_i$, $C_{i,j}$ is the conjunction of the fluent literals that appear in $P_{i,j}$.

From $\mathcal{K}_a$ to Update Axioms Since sensing does not affect the world state, the state update axiom of a sensing action a is independent of the knowledge bag of this action:

$$(\forall s)\, State(Do(a, s)) = State(s)$$

The knowledge update axiom for sensing action a is determined by knowledge bag $\mathcal{K}_a = \{(f_1, \varphi_1), \ldots, (f_m, \varphi_m)\}$ as follows:

$$\begin{array}{l} (\forall s)(\forall z)(KState(Do(a, s), z) \equiv \\ \qquad\qquad (KState(s, z) \wedge \\ [HOLDS(\varphi_1, s) \supset (Holds(f_1, z) \equiv Holds(f_1, s))] \wedge \\ [Unknown(f_1, s) \supset \\ \qquad\qquad (HOLDS(\varphi_1, z) \equiv HOLDS(\varphi_1, s))] \\ \wedge \ldots \wedge \\ [HOLDS(\varphi_m, s) \supset (Holds(f_m, z) \equiv Holds(f_m, s))] \wedge \\ [Unknown(f_m, s) \supset \\ \qquad\qquad (HOLDS(\varphi_m, z) \equiv HOLDS(\varphi_m, s))])) \end{array}$$

A side-effect of sensing in $\mathcal{A}_k$ is that the conditions of a sensing action become known to the agent, if the potential sensing effect of this action is previously unknown. To reflect this, we have defined the knowledge update axiom in such a way that a sensing action's precondition φ of a potential sensing effect f will be known in the successor situation if f is previously unknown.

Example (continued) The knowledge bag of the sensing action *Look* of D_1 is $\mathcal{K}_{Look} = \{\ (DrOpn, FacingDr)\}$. Then as the state and knowledge update axiom for *Look* we obtain, respectively, $(\forall s)\, State(Do(Look, s)) = State(s)$ and axiom (9) mentioned earlier.

5 Query Translation

The query translation function Π_Q translates an $\mathcal{A}_k$ query into a fluent calculus formula. The translation uses the recursive function τ that maps any $\mathcal{A}_k$ plan into a formula in fluent calculus.

Definition 12 *Let α, α_1, α_2 be $\mathcal{A}_k$ plans, a be an action, and ℓ be a fluent literal. The plan translation function τ is a function such that for any fluent calculus situation constant S_i, where S_{Final} denotes the final situation reached after executing the plan,*[7]

$$\tau([\,], S_i) \stackrel{\text{def}}{=} [\ S_i = S_{Final}\ ]$$

$$\tau([a|\alpha], S_i) \stackrel{\text{def}}{=} [\ S_{i+1} = Do(a, S_i)\ ]\ \wedge\ \tau(\alpha, S_{i+1})$$

$$\begin{aligned}\tau([\textbf{if}\ \ell\ \textbf{then}\ \alpha_1|\alpha], S_i) &\stackrel{\text{def}}{=}\\ &[\,(Knows(\ell, S_i) \supset \tau(\alpha_1;\alpha, S_i))\ \wedge\\ &\ (Knows(\neg\ell, S_i) \supset \tau(\alpha, S_i))\ ]\end{aligned}$$

$$\begin{aligned}\tau([\textbf{if}\ \ell\ \textbf{then}\ \alpha_1\ \textbf{else}\ \alpha_2|\alpha], S_i) &\stackrel{\text{def}}{=}\\ &[\,(Knows(\ell, S_i) \supset \tau(\alpha_1;\alpha, S_i))\ \wedge\\ &\ (Knows(\neg\ell, S_i) \supset \tau(\alpha_2;\alpha, S_i))\ ]\end{aligned}$$

*Given an $\mathcal{A}_k$ query of the form 'ℓ **after** α',*

$$\Pi_Q(\ell\ \textbf{after}\ \alpha) \stackrel{\text{def}}{=} \tau(\alpha, S_0) \supset Knows(\ell, S_{Final})$$

Example (continued) Query (5) is translated as follows:

$$\tau([Look, \textbf{if}\ldots], S_0)$$

$$\begin{aligned}\equiv\ & S_1 = Do(Look, S_0)\ \wedge\\ & \tau([\textbf{if}\ \neg DrOpn\ \textbf{then}\ [SendId]], S_1)\end{aligned}$$

$$\begin{aligned}\equiv\ & S_1 = Do(Look, S_0)\ \wedge\\ &\quad [\ Knows(\neg DrOpn, S_1) \supset \tau([SendId], S_1)]\ \wedge\\ &\quad [Knows(\neg\neg DrOpn, S_1) \supset \tau([\,], S_1)]\end{aligned}$$

$$\begin{aligned}\equiv\ & S_1 = Do(Look, S_0)\ \wedge\\ & [\ Knows(\neg DrOpn, S_1) \supset S_2 = Do(SendId, S_1)\ \wedge\\ &\qquad\qquad\qquad\qquad S_2 = S_{Final}]\\ & \wedge\ [Knows(DrOpn, S_1) \supset S_1 = S_{Final}]\end{aligned}$$

Let Δ be the resulting formula, then the translated query is

$$\Delta \supset Knows(DrOpn, S_{Final})$$

[7] Below, $\alpha_1;\alpha_2$ is a macro denoting the concatenation of two plans α_1 and α_2.

which can be shown to be a logical consequence of the fluent calculus axiomatization $\Pi(D_1)$ for our running example domain.

The following main result of our work says that fluent calculus is sound and complete wrt. the semantics of $\mathcal{A}_k$.

Theorem 1 *Given a consistent $\mathcal{A}_k$ domain description D, a plan α, and a fluent literal ℓ. Then*

$$D \models_{\mathcal{A}_k} \ell \textbf{ after } \alpha \quad \text{iff} \quad \Pi(D) \models \Pi_Q(\ell \textbf{ after } \alpha)$$

Proof (sketch) The 0-models coincide with the models for the translated update axioms for non-sensing actions, and the epistemic state resulting from performing a single sensing action coincides with the models for the successor knowledge state determined by the translated knowledge update axioms. By induction, this equivalence can be generalized to sequences of actions and complex plans.

Space restrictions do not permit us to give the complete proof of this theorem; we refer to [5].

6 Query Answering in $\mathcal{A}_k$ using FLUX

The programming language FLUX is a recent implementation of fluent calculus based on Constraint Logic Programming [13]. Its distinguishing feature is to support incomplete states, whereby negative and disjunctive information is encoded by constraints. The kernel of FLUX, which includes a declarative constraint solver, has been formally verified against the foundational axioms of fluent calculus [14]. We have extended this kernel by an implementation of our translation function, mapping an $\mathcal{A}_k$ domain descriptions into a FLUX program and an $\mathcal{A}_k$ query into a FLUX query. As a result, FLUX provides a query answering mechanism for the Action Description Language $\mathcal{A}_k$. A complementary use of the translation function can be to employ $\mathcal{A}_k$ as a high-level surface language for specifying action domains in FLUX when this system is used as a high-level programming method for cognitive agents that reason about their actions and plan [13].

Both the translation function as well as the FLUX kernel along with some examples are available at `www.cl.inf.tu-dresden.de/~ozan/ papers.html`.

7 Discussion

In being correct with respect to full $\mathcal{A}_k$ except for queries including loops, fluent calculus, as well as FLUX, is more expressive than most existing systems for reasoning about actions and planning with sensing actions, such as [4, 1, 8, 3]. These approaches use restricted notions of incomplete states, which do not allow for handling any kind of disjunctive information or reasoning about cases as required, for example, if an action is described in $\mathcal{A}_k$ to have conditional effects depending on whether some unknown fluent is true or false, but where a query can be proved under both conditional effects [1]. A general solution to the frame problem for knowledge is realized in the systems [6, 10], both of which are based on GOLOG [7]. However, an important restriction of these systems compared to the underlying situation calculus, is that they do not provide ways of actually deriving whether something is known after a sequence of actions.

Future work will include to extend our translation to cover loops in $\mathcal{A}_k$ queries. While a loop can be easily formalized by a second-order closure axiom in fluent calculus, it remains an open question how the effect of a loop can be actually inferred in a logic programming system like FLUX.[8]

References

1. C. Baral and T. Son. Approximate reasoning about actions in presence of sensing and incomplete information. In J. Maluszynski, ed., *Proc. of ILPS*, 387–401, 1997. MIT Press.
2. M. Gelfond and V. Lifschitz. Representing action and change by logic programs. *J. of Log. Prog.*, 17:301–321, 1993.
3. Giuseppe De Giacomo, Luca Iocchi, Daniele Nardi, and Riccardo Rosati. Planning with sensing for a mobile robot. In *Proc. of ECP*, vol. 1348 of *LNAI*, 158–170. Springer 1997.
4. K. Golden and D. Weld. Representing sensing actions: The middle ground revisited. In L. C. Aiello, J. Doyle, and S. Shapiro, ed.s, *Proc. of KR*, 174–185, 1996.
5. O. Kahramanoğulları. A translation from the action description language $\mathcal{A}_k$ to the fluent calculus. Master's thesis, Department of Computer Science, Dresden University of Technology, 2002. URL: `www.cl.inf.tu-dresden.de/ozan/papers.html`.
6. G. Lakemeyer. On sensing and off-line interpreting GOLOG. In H. Levesque and F. Pirri, ed.s, *Logical Foundations for Cognitive Agents*, 173–189. Springer, 1999.
7. H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *J. of Log. Prog.*, 31(1–3):59–83, 1997.
8. J. Lobo. COPLAS: A conditional planner with sensing actions. In *Cognitive Robotics*, vol. FS–98–02 of *AAAI Fall Symposia*, 109–116. AAAI Press 1998.
9. J. Lobo, G. Mendez, and S. Taylor. Knowledge and the action description language $\mathcal{A}$. *Theory and Practise of Log. Prog.*, 1(2):129–184, 2001.
10. R. Reiter. On knowledge-based programming with sensing in the situation calculus. *ACM Transactions on Computational Logic*, 2(4):433–457, 2001.
11. E. Sandewall. *Features and Fluents. The Representation of Knowledge about Dynamical Systems.* Oxford University Press, 1994.
12. M. Thielscher. Representing the knowledge of a robot. In A. Cohn, F. Giunchiglia, and B. Selman, ed.s, *Proc. of KR*, 109–120, 2000. Morgan Kaufmann.
13. M. Thielscher. Programming of reasoning and planning agents with FLUX. In D. Fensel, D. McGuinness, and M.-A. Williams, ed.s, *Proc. of KR*, 435–446, 2002. Morgan Kaufmann.
14. M. Thielscher. Reasoning about actions with CHRs and finite domain constraints. In P. Stuckey, ed., *Proc. of ICLP*, vol. 2401 of *LNCS*, 70–84, 2002. Springer.

[8] The concept of loops in GOLOG is too restricted to this end, because GOLOG supports nondeterministic choice of actions but not the specification of actions that have nondeterministic effects, as in $\mathcal{A}_k$.