A Unifying Action Calculus

Michael Thielscher*

School of Computer Science and Engineering, The University of New South Wales, Australia

Abstract

McCarthy's Situation Calculus is arguably the oldest special-purpose knowledge representation formalism, designed to axiomatize knowledge of actions and their effects. Four decades of research in this area have led to a variety of alternative formalisms: While some approaches can be considered instances or extensions of the classical Situation Calculus, like Reiter's successor state axioms or the Fluent Calculus, there are also special planning languages like ADL and approaches based on a linear (rather than branching) time structure like the Event Calculus. The co-existence of many different calculi has two main disadvantages: The formal relations among them is a largely open issue, and a lot of today's research concerns the transfer of specific results from one approach to another. In this paper, we present a unifying action calculus, which encompasses (well-defined classes of) all of the aforementioned formalisms. Our calculus not only facilitates comparisons and translations between specific approaches, it also allows to solve interesting problems for various calculi at once. We exemplify this by providing a general, calculus-independent solution to a problem of practical relevance, which is intimately related to McCarthy's quest for elaboration tolerant formalisms: the modularity of domain axiomatizations.

Key words: Knowledge Representation, Reasoning About Actions, Situation Calculus

1. Introduction

John McCarthy's Situation Calculus [22] is arguably the oldest special-purpose knowledge representation formalism. The aim is to use classical logic to axiomatize knowledge of actions and their effects. This is relevant for a variety of areas in AI, including planning, intelligent agents, high-level cognitive robotics, natural language understanding, and general game playing. While the Situation Calculus is the classical approach for this purpose, a variety of different logic-based formalisms have emerged in the course of the past decades, motivated mainly by the fundamental Frame Problem [25]. Besides prominent variants of the Situation Calculus like Reiter's successor state axioms [31] or the Fluent Calculus [41], planning languages like STRIPS, ADL, and PDDL [5, 29, 26] have been developed, which allow for simple operational solutions to the Frame Problem at the expense of a significantly limited expressiveness. Furthermore, the underlying branching time structure of the Situation Calculus has been replaced by a linear time structure in the Event Calculus and a number of other approaches [18, 36, 4, 10]. The basic principles of knowledge representation for actions are also used in special-purpose formalisms like the Game Description Language [8].

The co-existence of a multitude of knowledge representation languages for actions has two significant consequences for the research in this area. Firstly, there is a growing need both for comparative analysis of the expressiveness of different approaches as well as for translations from one specific language into another one. Previous studies along this line are [17, 28, 35, 3], each of which concerns the comparison of two specific formalisms. However, a method that encompasses a wide variety of alternative formalisms at the same time may allow for a more uniform way of assessing and translating calculi. Secondly, issues of general interest

*Corresponding author
Email addresses: mit@cse.unsw.edu.au (Michael Thielscher)

need to be separately addressed within each individual language. This often leads to a multiplication of research efforts. A notorious example is the Ramification Problem, that is, the problem of determining the indirect effects of actions [9], for which a variety of individual solutions have been developed for different formalisms, e.g., [19, 21, 11, 40, 38, 27]. A general method which enables a uniform treatment of problems across different calculi would help to avoid this multiplication of research efforts.

In this paper, we address both of these issues at the same time by proposing a unifying action calculus, which is independent of a specific solution to the Frame Problem and which is shown to be general enough to encompass a variety of different action representation formalisms. Most notably, it abstracts from the underlying time structure (branching or linear) and thus can be instantiated with both Situation Calculus-style approaches as well as Event Calculus-like languages. In so doing, our general calculus provides a uniform method for translating a variety of specific formalisms into each other. Moreover, the unifying approach allows to abstract from specific formalisms when investigating problems of general interest. We exemplify this by providing a new, calculus-independent solution to a problem of practical relevance for any action representation language: the modularity of domain axiomatizations [13]. Our result is a contribution to McCarthy's quest for elaboration tolerant formalisms [24], since modularity is a prerequisite for elaboration tolerance: theories with a variety of dependencies among different parts may not allow for the addition of new information without disrupting the entire axiomatization [14]. We use our unifying action calculus to develop a general method for verifying that a given set of domain constraints, precondition axioms, and effect formulas is free of undesired, implicit dependencies. We exemplify the range of applicability of this result by instantiating it for several specific approaches, in particular the Situation-, Fluent-, and Event Calculus.

The remainder of this paper is organized as follows. In the next section, we formally define an action calculus which abstracts from a specific underlying time structure and is independent of a specific solution to the Frame Problem. We illustrate the expressiveness of our definition by formalizing several example domains known from the literature, including nondeterministic actions, indirect effects, and actions with duration. In Section 3, we show how our unifying calculus can be used as an intermediary language for translations between specific languages. Specifically, we present two new results: a translation from ADL planning problems into the Event Calculus and a translation from the basic Fluent Calculus into a new extension—suitable for nondeterministic actions—of Reiter's basic Situation Calculus. In the second part of the paper, in Section 4, we show how the unifying action calculus can be used to provide a calculus-independent solution to the problem of implicit dependencies among domain axioms, and we again exemplify the range of applicability of this result by instantiating it for several action formalisms. We conclude with a discussion in Section 5.

2. A Unifying Action Calculus

The purpose of this section is to develop a unifying action calculus that abstracts from a variety of existing axiomatization techniques for describing actions and change. Logic-based action representation formalisms have in common two fundamental elements: *Fluents* [22] (sometimes called *features* [33]) represent properties of the domain that may change in response to the execution of *actions* (or *events* [18]). Fluents and actions are therefore basic sorts in the sorted logic language we are going to define. Action calculi also need to distinguish different points in time in order to axiomatize the changes caused by actions. We assume an abstract notion of time—which may be linear or branching—as the third fundamental sort.

The three basic sorts are used for three fundamental predicates: The relation $t_1 < t_2$ denotes a (possibly partial) ordering on the time structure. Predicate $Holds(f, t)$ is used to say that fluent f is true at time t. Finally, the intended meaning of expression $Poss(a, s, t)$ is that it is possible to do action a beginning at time s and ending at time t. These three predicates, along with the three fundamental sorts, form the basis of a domain signature in our unifying action calculus.

Definition 1. A *domain signature* is a finite, sorted logic language which includes the sorts FLUENT,

ACTION, and TIME along with the predicates

$$\begin{aligned} <&: \text{TIME} \times \text{TIME} \\ \mathit{Holds}&: \text{FLUENT} \times \text{TIME} \\ \mathit{Poss}&: \text{ACTION} \times \text{TIME} \times \text{TIME} \end{aligned}$$

We tacitly assume that a signature always includes the standard predicate "=", interpreted as true equality. As usual, then, $s \leq t$ stands for $s < t \vee s = t$. □

Throughout the paper we will denote variables of sort ACTION by the letter a, variables of sort FLUENT by f and g, and variables of sort TIME by s and t. We tacitly assume uniqueness-of-names [1] for all functions into FLUENT and ACTION, which is a common assumption in all standard action calculi.

Next, we define the notion of a state formula, which allows to express properties of a domain at given times.

Definition 2. Let $\vec{t}$ be a non-empty sequence of variables of sort TIME in a given domain signature. A *state formula in* $\vec{t}$ is a first-order formula $\Phi[\vec{t}]$ in which the variables in $\vec{t}$ occur free and such that

1. for each occurrence of $\mathit{Holds}(f,t)$ in Φ we have $t \in \vec{t}$;
2. predicate Poss does not occur in Φ.

□

Similar notions are used in many existing calculi but usually restricted to a single time point. As will be shown later in this section, the more general concept is useful, for instance, when axiomatizing actions with ramifications.

We are now in a position to formalize, in our calculus, three fundamental categories of domain axioms: domain constraints, which describe state properties that hold at all times; precondition axioms, which define the conditions for actions to be applicable in a state; and effect axioms, which define the consequences of actions. For the latter, we use a general form that allows to define nondeterministic actions with the help of different possible "cases" $i = 1, \ldots, k$ of updates $\Upsilon_i[s,t]$ (cf. axiom (1) below). Each of these sub-formulas defines the fluents that hold after the action, at time t, relative to the state when the action starts, at time s. This does not only concern all (possibly conditional) effects of an action but also all non-effects. This formulation is general enough to subsume specific solutions to the Frame and Ramification Problem.

Definition 3. Consider a domain signature, and let A be a function into sort ACTION.

1. A *domain constraint* is of the form
$$\delta[t]$$
which is a state formula in t.[1]
2. A *precondition axiom* is of the form
$$\mathit{Poss}(A(\vec{x}), s, t) \equiv \pi_A[s]$$
where $\pi_A[s]$ is a state formula in s with free variables among $s, t, \vec{x}$.
3. An *effect axiom* is of the form
$$\mathit{Poss}(A(\vec{x}), s, t) \supset \Upsilon_1[s,t] \vee \ldots \vee \Upsilon_k[s,t] \tag{1}$$
where $k \geq 1$ and each $\Upsilon_i[s,t]$ $(1 \leq i \leq k)$ is a formula of the form
$$\begin{aligned} (\exists \vec{y}_i)(\Phi_i[s] \wedge (\forall f)\,[\Gamma_i^+[s,t] \supset \mathit{Holds}(f,t)] \\ \wedge\ (\forall f)\,[\Gamma_i^-[s,t] \supset \neg \mathit{Holds}(f,t)]) \end{aligned} \tag{2}$$
in which $\Phi_i[s]$ is a state formula in s with free variables among $s, \vec{x}, \vec{y}$,[2] and both $\Gamma_i^+[s,t]$ and $\Gamma_i^-[s,t]$ are state formulas in s,t with free variables among $f, s, t, \vec{x}, \vec{y}$.

[1] Throughout the paper, free variables are assumed universally quantified.

[2] The purpose of sub-formula $\Phi_i[s]$ is to define conditions for case i to apply. Whenever it is tautology, we will simply omit this formula.

A *domain axiomatization* consists of precondition and effect axioms, one each for every function into sort ACTION, along with a finite set of domain constraints and a finite set of *foundational axioms* without the predicates *Holds* and *Poss*. □

The purpose of the foundational axioms is to define the underlying time structure. In the following we present example axiomatizations of several domains known from the literature to illustrate the wide range of phenomena that can be expressed in this unifying calculus.

Example 1. (branching time, nondeterministic actions)

The Situation Calculus and related axiomatization techniques are based on a branching time-structure, where the *situations* are commonly defined by the constant S_0 : TIME and the function Do : ACTION × TIME ↦ TIME. A standard example for a nondeterministic action is that of dropping a pin on a checkerboard [20]. The pin may land on a white square, a black square, or both. Let the fluents $\texttt{Pin}(x)$, $\texttt{White}(x)$, and $\texttt{Black}(x)$ denote that x is a pin and that it is, respectively, on a white and a black square. The action $\texttt{Drop}(x)$ can then be axiomatized by the following precondition and effect axiom.

$$\begin{aligned} Poss(\texttt{Drop}(x), s, t) \equiv\ & Holds(\texttt{Pin}(x), s) \wedge \neg Holds(\texttt{White}(x), s) \wedge \neg Holds(\texttt{Black}(x), s) \wedge \\ & t = Do(\texttt{Drop}(x), s) \end{aligned} \tag{3}$$

$$\begin{aligned} Poss(\texttt{Drop}(x), s, t) \supset\ & ((\forall f)\,[f = \texttt{White}(x) \vee Holds(f, s) \supset Holds(f, t)] \wedge \\ & (\forall f)\,[f \neq \texttt{White}(x) \wedge \neg Holds(f, s) \supset \neg Holds(f, t)]) \\ & \vee \\ & ((\forall f)\,[f = \texttt{Black}(x) \vee Holds(f, s) \supset Holds(f, t)] \wedge \\ & (\forall f)\,[f \neq \texttt{Black}(x) \wedge \neg Holds(f, s) \supset \neg Holds(f, t)]) \\ & \vee \\ & ((\forall f)\,[f = \texttt{White}(x) \vee f = \texttt{Black}(x) \vee Holds(f, s) \supset Holds(f, t)] \wedge \\ & (\forall f)\,[f \neq \texttt{White}(x) \wedge f \neq \texttt{Black}(x) \wedge \neg Holds(f, s) \supset \neg Holds(f, t)]) \end{aligned} \tag{4}$$

The three disjuncts in the effect axiom encode the three possible outcomes of the nondeterministic action, including the case where the pin lands on both white and black. Each of the sub-formulas describing the update includes a solution to the Frame Problem: all fluents except possibly $\texttt{Black}(x)$ and $\texttt{White}(x)$ retain their value in the successor situation $t = Do(\texttt{Drop}(x), s)$. As an example, suppose given the initial state description,

$$Holds(\texttt{Pin}(\texttt{A}), S_0) \wedge \neg Holds(\texttt{White}(\texttt{A}), S_0) \wedge \neg Holds(\texttt{Black}(\texttt{A}), S_0) \tag{5}$$

along with the observation $\neg Holds(\texttt{White}(\texttt{A}), Do(\texttt{Drop}(\texttt{A}), S_0))$. Together with the domain axioms this implies that $Holds(\texttt{Black}(\texttt{A}), Do(\texttt{Drop}(\texttt{A}), S_0))$. To see why, from (5) and precondition axiom (3) it follows that $Poss(\texttt{Drop}(\texttt{A}), S_0, Do(\texttt{Drop}(\texttt{A}), S_0))$. The first and third disjunct in effect axiom (4) then imply $Holds(\texttt{White}(\texttt{A}), Do(\texttt{Drop}(\texttt{A}), S_0))$, which contradicts the observation. Therefore the claim follows by the second disjunct. □

It is worth mentioning that some nondeterministic actions can be formulated in the unifying calculus without a disjunctive effect axiom, namely by simply excluding one or more fluents from the frame assumption. An example is the following axiomatization of the effect of tossing a coin.

$$\begin{aligned} Poss(\texttt{Toss}(c), s, t) \supset\ & (\forall f)\,[f \neq \texttt{Heads}(c) \wedge Holds(f, s) \supset Holds(f, t)] \wedge \\ & (\forall f)\,[f \neq \texttt{Heads}(c) \wedge \neg Holds(f, s) \supset \neg Holds(f, t)] \end{aligned}$$

where fluent $\texttt{Heads}(c)$ denotes that coin c shows heads. This corresponds to the notion of *occlusion* or *release* as used in, e.g., [34] and [37].

Example 2. (branching time, ramifications)

Consider a variant, introduced in [2], of the famous Yale Shooting scenario [12] with fluents `Loaded`, `Alive`, and `Walking` representing that a gun is loaded and that the turkey is alive and walking, respectively. The following domain constraint says that the turkey can walk only if it is alive.

$$Holds(\texttt{Walking}, t) \supset Holds(\texttt{Alive}, t) \tag{6}$$

Let `Load`, `Wait`, and `Shoot` denote the actions of loading the gun, waiting, and shooting, respectively. Their preconditions shall be as follows.

$$\begin{array}{rcl} Poss(\texttt{Load}, s, t) & \equiv & t = Do(\texttt{Load}, s) \\ Poss(\texttt{Wait}, s, t) & \equiv & t = Do(\texttt{Wait}, s) \\ Poss(\texttt{Shoot}, s, t) & \equiv & t = Do(\texttt{Shoot}, s) \end{array} \tag{7}$$

The following schema for the effect axioms encodes a combined solution to the Frame and Ramification Problem which is a reformulation of the causal approach described in [10].

$$\begin{array}{rl} Poss(a, s, t) \supset & (\forall f)\,[CausedT(f, a, s, t) \vee \neg CausedF(f, a, s, t) \supset Holds(f, t)] \wedge \\ & (\forall f)\,[CausedF(f, a, s, t) \vee \neg CausedT(f, a, s, t) \supset \neg Holds(f, t)] \end{array} \tag{8}$$

where

$$\begin{array}{rcl} CausedT(f, a, s, t) & \stackrel{\text{def}}{=} & Holds(f, s) \wedge Holds(f, t) \vee \\ & & f = \texttt{Loaded} \wedge a = \texttt{Load} \\ & & \\ CausedF(f, a, s, t) & \stackrel{\text{def}}{=} & \neg Holds(f, s) \wedge \neg Holds(f, t) \vee \\ & & f = \texttt{Alive} \wedge Holds(\texttt{Loaded}, s) \wedge a = \texttt{Shoot} \vee \\ & & f = \texttt{Walking} \wedge \neg Holds(\texttt{Alive}, t) \end{array} \tag{9}$$

Macro $CausedT(f, a, s, t)$ combines a positive frame assumption (that is, $Holds(f, s) \wedge Holds(f, t)$) with the possible positive effects in the domain; here, the fact that the gun becomes loaded by loading it. Likewise, $CausedF(f, a, s, t)$ combines a negative frame assumption with the possible negative effects: if the gun is loaded when shooting it, then the turkey dies; and if in a successor situation the turkey is not alive, then this causes it not to be walking. The latter describes an indirect effect related to the domain constraint, (6), but conveying additional, causal information to solve the Ramification Problem [10]. (Note also that, after two decades of shooting a turkey in Yale, so doing no longer unloads the gun.) It is easy to see that action variable a in schema (8) can be instantiated by the three actions of this domain in order to obtain effect axioms that are actually of the form required by Definition 3.

The domain axiomatization entails, for example, $\neg Holds(\texttt{Walking}, Do(\texttt{Shoot}, Do(\texttt{Wait}, Do(\texttt{Load}, S_0))))$. To see why, let $S_1 = Do(\texttt{Load}, S_0)$, $S_2 = Do(\texttt{Wait}, S_1)$, and $S_3 = Do(\texttt{Shoot}, S_2)$. Precondition axioms (7) imply $Poss(\texttt{Load}, S_0, S_1)$, $Poss(\texttt{Wait}, S_1, S_2)$, and $Poss(\texttt{Shoot}, S_2, S_3)$. From definition (9), we obtain $CausedT(\texttt{Loaded}, \texttt{Load}, S_0, S_1)$; hence, effect axiom (8) entails

$$Holds(\texttt{Loaded}, S_1) \tag{10}$$

The gun remains loaded through the subsequent `Wait` action: given uniqueness-of-names, (9) implies that $CausedF(\texttt{Loaded}, \texttt{Wait}, S_1, S_2)$ is equivalent to $\neg Holds(\texttt{Loaded}, S_1) \wedge \neg Holds(\texttt{Loaded}, S_2)$. Because of (10), this actually implies $\neg CausedF(\texttt{Loaded}, \texttt{Wait}, S_1, S_2)$. It follows from (8) that

$$Holds(\texttt{Loaded}, S_2) \tag{11}$$

(It is worth mentioning that this, together with (10), implies $CausedT(\texttt{Loaded}, \texttt{Wait}, S_1, S_2)$. This is how the Frame Problem is solved in the axiomatization technique used here: fluent `Loaded` is "caused" to be true in S_2 simply by staying true from S_1 to S_2.) Given that the gun is still loaded, the `Shoot` action has the effect that the turkey dies and, as an indirect effect, that it stops walking: from (9) and (11) it follows that $CausedF(\texttt{Alive}, \texttt{Shoot}, S_2, S_3)$. Hence, $\neg Holds(\texttt{Alive}, S_3)$ by (8). This, in turn, implies $CausedF(\texttt{Walking}, \texttt{Shoot}, S_2, S_3)$ according to (9), from which the claim follows by (8) again. □

Example 3. (linear time, actions with duration)

The Event Calculus and other axiomatization techniques use a linear time structure, like for example the natural numbers. The following scenario is adapted from [33]. Let the fluents `Assembled` and `Instr` denote, respectively, the state of an assembly kit and the availability of assembly instructions. To represent

the occurrence of actions, we add a fluent called $\texttt{Occurs}(a, s, t)$, which describes the fact that action a actually happens, starting at time s and ending at time t. We consider two actions: `Assemble`, which has the effect that the kit is assembled in the end; and the special action `Inert`, which is used to axiomatize the frame assumption between two time points when nothing else happens:

$$\begin{aligned} \mathit{Poss}(\texttt{Assemble}, s, t) \supset\ & (\forall f)\,[f = \texttt{Assembled} \vee \mathit{Holds}(f, s) \supset \mathit{Holds}(f, t)] \wedge \\ & (\forall f)\,[f \neq \texttt{Assembled} \wedge \neg\mathit{Holds}(f, s) \supset \neg\mathit{Holds}(f, t)] \\ \mathit{Poss}(\texttt{Inert}, s, t) \supset\ & (\forall f)\,[\mathit{Holds}(f, s) \supset \mathit{Holds}(f, t)] \wedge \\ & (\forall f)\,[\neg\mathit{Holds}(f, s) \supset \neg\mathit{Holds}(f, t)] \end{aligned} \tag{12}$$

The preconditions for the two actions are as follows.

$$\begin{aligned} \mathit{Poss}(\texttt{Assemble}, s, t) \equiv\ & \mathit{Holds}(\texttt{Occurs}(\texttt{Assemble}, s, t), s) \wedge \\ & [\mathit{Holds}(\texttt{Instr}, s) \supset t = s + 20] \wedge [\neg\mathit{Holds}(\texttt{Instr}, s) \supset t = s + 60] \\ \mathit{Poss}(\texttt{Inert}, s, t) \equiv\ & \neg(\exists a, s', t')\,(\mathit{Holds}(\texttt{Occurs}(a, s', t'), s) \wedge s' < t \wedge t' > s) \end{aligned} \tag{13}$$

Put in words, assembling has a variable duration, depending on the availability of instructions. The generic `Inert` action is possible between any two time points s and t whenever there is no action that starts before t and ends after s. In order that the second precondition axiom in (13) complies with Definition 3, the right-hand side must be a state formula in s. This, in turn, requires all action occurrences to be "known" at all times, which can be easily obtained by the generic definition of domain constraints via the following macro.

$$\mathit{Happens}(a, s, t) \stackrel{\text{def}}{=} (\forall t')\,\mathit{Holds}(\texttt{Occurs}(a, s, t), t') \tag{14}$$

As an example, consider a scenario in which only a single action occurs,

$$(\exists t_a)\,(\mathit{Poss}(\texttt{Assemble}, 100, t_a) \wedge (\forall a, s, t)\,[\mathit{Happens}(a, s, t) \equiv a = \texttt{Assemble} \wedge s = 100 \wedge t = t_a]\,) \tag{15}$$

along with the observation $\neg\mathit{Holds}(\texttt{Assembled}, 130)$. This, together with the domain axioms, implies $\neg\mathit{Holds}(\texttt{Instr}, 0)$; that is to say, if the kit is not yet assembled at time 130 then the instructions were not present from the beginning. To see why, suppose, for the sake of argument, that $\mathit{Poss}(\texttt{Assemble}, 100, 120)$. We show that this leads to a contradiction: from (12) it follows that

$$\mathit{Holds}(\texttt{Assembled}, 120) \tag{16}$$

Moreover, (15), definition (14), and $\mathit{Poss}(\texttt{Assemble}, 100, 120)$ along with (13) imply that there is no instance $\mathit{Holds}(\texttt{Occurs}(a, s', t'), 120)$ such that $t' > 120$. Therefore, $\mathit{Poss}(\texttt{Inert}, 120, 130)$ according to (13). In turn, this and (16) imply $\mathit{Holds}(\texttt{Assembled}, 130)$ by (12), which contradicts the observation. Hence, $\neg\mathit{Poss}(\texttt{Assemble}, 100, 120)$. This and $\mathit{Poss}(\texttt{Assemble}, 100, t_a)$ entail $\neg\mathit{Holds}(\texttt{Instr}, 100)$ due to (13). The claim $\neg\mathit{Holds}(\texttt{Instr}, 0)$ then follows from (12) and the fact that $\mathit{Poss}(\texttt{Inert}, 0, 100)$ (which the reader may easily verify from (14) and (15)). □

These examples show that the unifying calculus allows to model a variety of ontological features. We conclude this section by defining three important sub-classes of domain axiomatizations, which will be used later in the paper. To begin with, a domain axiomatization is called *progressing* if there is a least time point and if time always moves forward when an action is executed.

Definition 4. A domain axiomatization with precondition axioms Π and foundational axioms Ω is *progressing* if

1. $\Omega \models (\exists s : \textsc{time})\,(\forall t : \textsc{time})\,s \leq t$ and
2. $\Pi \cup \Omega \models \mathit{Poss}(a, s, t) \supset s < t$.

□

A domain axiomatization is called *sequential* if it is progressing and no two actions overlap.

Definition 5. A domain axiomatization with precondition axioms Π and foundational axioms Ω is *sequential* if it is progressing and

$$\Pi \cup \Omega \models Poss(a,s,t) \wedge Poss(a',s',t') \supset (t < t' \supset t \leq s') \wedge (t = t' \supset a = a' \wedge s = s')$$

□

In Example 2 we have seen how ramifications of actions are obtained if an effect depends on conditions of the successor state, as in

$$f = \texttt{Walking} \wedge \neg Holds(\texttt{Alive}, t) \supset \neg Holds(f, t)$$

(cf. (9) and (8), respectively). This gives rise to the following definition.

Definition 6. A domain axiomatization is *ramification-free* if each $\Upsilon_i[s,t]$ in an effect axiom (1) is of the form

$$\begin{aligned}(\exists \vec{y}_i)(\Phi_i[s] \wedge (\forall f)\,[\Gamma_i^+[s] \supset Holds(f,t)] \\ \wedge (\forall f)\,[\Gamma_i^-[s] \supset \neg Holds(f,t)])\end{aligned}$$

where both $\Gamma_i^+[s]$ and $\Gamma_i^-[s]$ are state formulas in s. □

3. Translations Based on the Unifying Approach

In this section, we show how the unifying calculus can be used as an intermediary language for translating specific calculi into each other. The general idea is to map domain descriptions in one language into domain axiomatizations in the unifying calculus, and then to re-write those into the target language. This provides a uniform method for embedding (well-defined classes) of domains in a variety of existing approaches, which also allows to compare their relative expressiveness. The advantage of using the unifying action calculus as a middle language is two-fold:

1. A mapping from the source language into the unifying calculus results in a generic representation, which can then be further mapped onto a variety of different target languages, thus avoiding the need for complete translations in each case.
2. Once a mapping into a target language has been developed for a specific class of domains in the unifying calculus, it suffices to map a source language into that class to obtain a full translation from source to target.

We exemplify our method by two new results: an axiomatic characterization of ADL planning problems in the Event Calculus, and a translation of the basic Fluent Calculus into the Situation Calculus. As a by-product we define an extension of Reiter's basic Situation Calculus which is suitable for nondeterministic actions and which is somewhat more general than the approach proposed in [20].

3.1. Translating ADL into the Event Calculus

3.1.1. From ADL ...

In order to give an Event Calculus characterization of the planning language ADL, introduced in [29], we first interpret ADL in our unifying calculus. For the definition of this planning language we follow [6].

Definition 7. An *ADL signature* consists of: a finite set of types, possibly including definitions of types as disjunctive unions of other types; a finite set of typed constants; a finite set of typed fluents; and a finite set of typed action names.

An *ADL planning problem* is composed of the following elements.

1. For each operator name $A(\vec{x})$ a *precondition* π, which is a first-order formula with free variables among $\vec{x}$ and whose atoms are fluents $F(\vec{t})$ or equalities $t_1 = t_2$.

2. For each operator name $A(\vec{x})$ a finite set of *effect specifications*, which are of either of the forms

$$(\forall \vec{y}_i)\,(\gamma_i^+ \;\Rightarrow\; F_i(\vec{y}_i)) \quad \text{or} \quad (\forall \vec{y}_j)\,(\gamma_j^- \;\Rightarrow\; \neg F_j(\vec{y}_j)) \tag{17}$$

   where the conditions γ_i^+ and γ_j^- are first-order formulas with free variables among, respectively, $\vec{x}, \vec{y}_i$ and $\vec{x}, \vec{y}_j$, and whose atoms are fluents $F(\vec{t})$ or equalities $t_1 = t_2$.[3]
3. A conjunction of ground fluent literals as the (possibly incomplete) *initial state specification* and a closed first-order formula as the *planning goal*.

□

Example 4. Consider the following specification of the action $\texttt{Move}(\texttt{object}, \texttt{old}, \texttt{new})$ in the well-known blocksworld:

$$\begin{aligned}
\text{Precondition:}\quad & \texttt{On}(\texttt{object}, \texttt{old}) \wedge \neg(\texttt{old} = \texttt{new}) \wedge \\
& \neg(\exists z)\,\texttt{On}(z, \texttt{object}) \wedge \neg(\exists z)\,\texttt{On}(z, \texttt{new}) \\
\\
\text{Effects:}\quad & (\forall x, y)\,(x = \texttt{object} \wedge y = \texttt{new} \;\Rightarrow\; \texttt{On}(x, y)) \\
& (\forall x, y)\,(x = \texttt{object} \wedge y = \texttt{old} \;\Rightarrow\; \neg\texttt{On}(x, y)) \\
& (\forall x, y)\,(x = \texttt{object} \wedge y = \texttt{new} \;\Rightarrow\; \texttt{Above}(x, y)) \\
& (\forall x, z)\,(x = \texttt{object} \wedge \texttt{Above}(\texttt{new}, z) \;\Rightarrow\; \texttt{Above}(x, z)) \\
& (\forall x, z)\,(x = \texttt{object} \wedge \texttt{Above}(x, z) \;\Rightarrow\; \neg\texttt{Above}(x, z))
\end{aligned} \tag{18}$$

The bottommost two expressions specify an unbounded number of effects, which is one of the characteristic features of ADL that go beyond the expressiveness of STRIPS. □

The semantics of an ADL domain description requires the definition of a transition function on complete states. These are represented as ground sets of fluents S, and the basis for entailment is the definition $S \models F(\vec{t})$ iff $F(\vec{t}) \in S$ for a ground fluent $F(\vec{t})$.

Definition 8. Consider an ADL planning problem, and let $\mathcal{S}$ and $\mathcal{S}'$ be two sets of ground fluents and $A(\vec{t})$ a ground instance of an action with precondition π and effect specifications (17), then

$$\mathcal{S} \;\rightsquigarrow_{A(\vec{t})}\; \mathcal{S}'$$

if the action is *applicable* in $\mathcal{S}$, that is, $\mathcal{S} \models \pi\{\vec{x} \mapsto \vec{t}\}$, and if $\mathcal{S}' = \mathcal{S} \setminus \mathcal{D} \cup \mathcal{A}$, where the *delete-list* $\mathcal{D}$ is the set of all ground fluents $F_j(\vec{r})$ such that $\mathcal{S} \models \gamma_j^-\{\vec{x} \mapsto \vec{t}, \vec{y}_j \mapsto \vec{r}\}$ and the *add-list* $\mathcal{A}$ is the set of all ground fluents $F_i(\vec{r})$ such that $\mathcal{S} \models \gamma_i^+\{\vec{x} \mapsto \vec{t}, \vec{y}_i \mapsto \vec{r}\}$.

A sequence $\alpha_1, \ldots, \alpha_n$ of ground actions is a *solution* to the planning problem if for every $\mathcal{S}_0$ which satisfies the initial state,

$$\mathcal{S}_0 \;\rightsquigarrow_{\alpha_1}\; \ldots \;\rightsquigarrow_{\alpha_n}\; \mathcal{S}_n$$

such that $\mathcal{S}_n$ entails the planning goal. □

Because it is not a purely logical axiomatization, an ADL description does not presuppose a particular time structure. This allows for various interpretations; for example, [3] uses branching time for an embedding of ADL in the Situation Calculus. In view of a translation into the Event Calculus, we map ADL planning problems into our unifying calculus using a linear time structure, specifically the natural numbers. To begin with, the type declarations are taken as a specification of domain sorts. The ADL operator names are mapped onto functions into sort ACTION, and the ADL fluents are mapped onto functions into sort FLUENT. Much like in Example 3, we add the special fluent $\texttt{Occurs}(a : \text{ACTION}, t : \mathbb{N})$ to represent the occurrence of an action at a specific time. An ADL domain description can then be translated into a domain axiomatization using our unifying calculus as follows.

[3] The original definition actually allows to partially instantiate the arguments of the fluents $F_i(\vec{y}_i)$ and $F_j(\vec{y}_j)$ in (17). For the sake of simplicity, we assume that this is equivalently represented by equations in γ_i^+ and γ_j^-, respectively.

1. The foundational axioms define the time structure to be the natural numbers $\mathbb{N}$ with least element 0.
2. The precondition axioms are,

$$Poss(A(\vec{x}), s, t) \equiv Holds(\texttt{Occurs}(A(\vec{x}), s), s) \wedge t = s + 1 \wedge \pi_A[s] \tag{19}$$

   where $\pi_A[s]$ is the ADL precondition for operator $A(\vec{x})$ but with every occurrence of a fluent ϕ replaced by $Holds(\phi, s)$.
3. The effect axioms are,

$$\begin{array}{l} Poss(A(\vec{x}), s, t) \supset (\forall f)\,[\,(\Gamma_A^+ \vee Holds(f, s) \wedge \neg\Gamma_A^-) \supset Holds(f, t)] \wedge \\ \qquad\qquad\quad (\forall f)\,[\,(\Gamma_A^- \vee \neg Holds(f, s) \wedge \neg\Gamma_A^+) \supset \neg Holds(f, t)] \end{array} \tag{20}$$

   where

$$\begin{array}{l} \Gamma_A^+ \stackrel{\text{def}}{=} \bigvee_i (\exists \vec{y}_i)\,(\gamma_i^+[s] \wedge f = F(\vec{y}_i)\,) \\ \Gamma_A^- \stackrel{\text{def}}{=} \bigvee_j (\exists \vec{y}_j)\,(\gamma_j^-[s] \wedge f = F(\vec{y}_j)\,) \end{array} \tag{21}$$

   Here, $\gamma_i^+[s]$ and $\gamma_j^-[s]$ are as γ_i^+ and γ_j^- in the ADL effect specifications for action $A(\vec{x})$ but with every fluent ϕ replaced by $Holds(\phi, s)$.
4. An initial state I is mapped to the formula $I[0]$ where every fluent ϕ is replaced by $Holds(\phi, 0)$.
5. A sequence of actions $\alpha_1, \ldots, \alpha_n$ is mapped onto the formula

$$(\forall t)\,(Holds(\texttt{Occurs}(a, s), t) \equiv a = \alpha_1 \wedge s = 0 \vee \ldots \vee a = \alpha_n \wedge s = n - 1) \tag{22}$$

Example 4. (Continued) Recall effect specifications (18). These are mapped onto the general effect axiom (20) for $\texttt{Move}(\texttt{object}, \texttt{old}, \texttt{new})$ with

$$\begin{array}{l} \Gamma^+_{\texttt{Move}} \stackrel{\text{def}}{=} f = \texttt{On}(\texttt{object}, \texttt{new}) \vee f = \texttt{Above}(\texttt{object}, \texttt{new}) \vee \\ \qquad\qquad (\exists z)\,(Holds(\texttt{Above}(\texttt{new}, z), s) \wedge f = \texttt{Above}(\texttt{object}, z)) \\ \Gamma^-_{\texttt{Move}} \stackrel{\text{def}}{=} f = \texttt{On}(\texttt{object}, \texttt{old}) \vee \\ \qquad\qquad (\exists z)\,(Holds(\texttt{Above}(\texttt{object}, z), s) \wedge f = \texttt{Above}(\texttt{object}, z)) \end{array} \tag{23}$$

□

The mapping of ADL into the unifying calculus can be easily proved correct under the assumption that the effect specifications (17) for every action are *consistent*, that is,

$$\models \neg(\bigvee_i \gamma_i^+ \wedge \bigvee_j \gamma_j^-) \tag{24}$$

where i and j range over all effect formulas for the same fluent.

Proposition 9. *Let Σ be the domain axiomatization obtained from a consistent ADL domain with goal G, then a sequence of actions $\alpha_1, \ldots, \alpha_n$ is a solution to the planning problem iff*

$$\Sigma \cup \{(22)\} \models G[n]$$

where $G[n]$ is G but with every fluent ϕ replaced by $Holds(\phi, n)$.

Proof: Let $\mathcal{S}, \mathcal{S}'$ be two sets of ground fluents, s a time point, and $A(\vec{t})$ a ground action. In the following, by $\mathcal{S}[s]$ we denote the conjunction of all atoms $Holds(\phi, s)$ for which $\phi \in \mathcal{S}$, plus all $\neg Holds(\phi, s)$ for ground fluents such that $\phi \notin \mathcal{S}$. This is possible because there are only finitely many fluent functions and object constants.

Since the fluent for action occurrences does not feature in the ADL effect specifications, effect axiom (20) implies that

$$Holds(\texttt{Occurs}(a, s), t) \supset (\forall t')\, Holds(\texttt{Occurs}(a, s), t')$$

From (19) it then follows that $A(\vec{t})$ is applicable in $\mathcal{S}$ iff

$$\Sigma \cup \{Holds(\texttt{Occurs}(A(\vec{t}), s), s)\} \models \mathcal{S}[s] \supset Poss(A(\vec{t}), s, s+1)$$

Under the consistency assumption (24), effect axiom (20) then implies that $\mathcal{S} \rightsquigarrow_{A(\vec{t})} \mathcal{S}'$ iff

$$\Sigma \cup \{Poss(A(\vec{t}), s, s+1)\} \models \mathcal{S}[s] \supset \mathcal{S}'[s+1]$$

since, by Definition 8, for any ground fluent we have $F_i(\vec{r}) \in \mathcal{A}$ iff $\mathcal{S} \models \gamma_i^+\{\vec{x} \mapsto \vec{t}, \vec{y} \mapsto \vec{r}\}$ iff $\mathcal{S}[s] \models \Gamma^+\{\vec{x} \mapsto \vec{t}, \vec{y} \mapsto \vec{r}\}$, and $F_j(\vec{r}) \in \mathcal{D}$ iff $\mathcal{S} \models \gamma_j^-\{\vec{x} \mapsto \vec{t}, \vec{y} \mapsto \vec{r}\}$ iff $\mathcal{S}[s] \models \Gamma^-\{\vec{x} \mapsto \vec{t}, \vec{y} \mapsto \vec{r}\}$. Hence, $F(\vec{t}) \in \mathcal{S} \setminus \mathcal{D} \cup \mathcal{A}$ iff $\{(20)\} \cup \{Poss(A(\vec{t}), s, s+1)\} \models \mathcal{S}[s] \supset Holds(F(\vec{t}), s+1)$, and, conversely, $F(\vec{t}) \notin \mathcal{S} \setminus \mathcal{D} \cup \mathcal{A}$ iff $\{(20)\} \cup \{Poss(A(\vec{t}), s, s+1)\} \models \mathcal{S}[s] \supset \neg Holds(F(\vec{t}), s+1)$. This shows that the precondition and effect axioms correctly encode the update of states in ADL according to Definition 8. The claim then follows by straightforward induction on n. ∎

3.1.2. ... to the Event Calculus

The Event Calculus uses a linear time structure, which allows us to adopt directly the natural numbers used in the axiomatization of an ADL planning problem from above. In the simple Event Calculus [37], the predicate $Happens(a, t)$ denotes the occurrence of an action (a.k.a. event) at a given time point. Effects of actions are axiomatized based on the predicates $Initiates(a, f, t)$ and $Terminates(a, f, t)$ representing, respectively, the initiation and termination of fluent f at time t by action a. The Frame Problem is then solved in two steps. First, by minimizing predicates *Initiates* and *Terminates* using circumscription [23], and then by applying the following foundational axioms, where $Init_P(f)$ and $Init_N(f)$ are used to specify positive and negative initial conditions.

$$\begin{aligned} Holds(f,t) \subset\ & Init_P(f) \wedge \neg Clipped(0,f,t) \vee \\ & (\exists a, s)\,(Happens(a,s) \wedge Initiates(a,f,s) \wedge s < t \wedge \neg Clipped(s,f,t)) \\ \neg Holds(f,t) \subset\ & Init_N(f) \wedge \neg Declipped(0,f,t) \vee \\ & (\exists a, s)\,(Happens(a,s) \wedge Terminates(a,f,s) \wedge s < t \wedge \neg Declipped(s,f,t)) \end{aligned} \tag{25}$$

$$\begin{aligned} Clipped(s,f,t) &\stackrel{\text{def}}{=} (\exists a, t')\,(Happens(a,t') \wedge Terminates(a,f,t') \wedge s < t' < t) \\ Declipped(s,f,t) &\stackrel{\text{def}}{=} (\exists a, t')\,(Happens(a,t') \wedge Initiates(a,f,t') \wedge s < t' < t) \end{aligned}$$

Put in words, a fluent holds at time t if it is true initially or is initiated by an earlier action and not terminated in between (definition *Clipped*). Conversely, a fluent does not hold at time t if it is false initially or terminated by an earlier action and not initiated in between (definition *Declipped*).

In the following we show how the domain axiomatization resulting from mapping an ADL planning problem into our unifying calculus can be translated into the simple Event Calculus. Generally, the translation of a domain axiomatization into a language based on a linear time structure would require to introduce a special "occurrence" fluent $\texttt{Occurs}(a, s)$ and to identify, in case of the simple Event Calculus with natural numbers, the predicate $Happens(a, s)$ with $Poss(a, s, s+1) \wedge Holds(\texttt{Occurs}(a, s), s)$. For the sake of simplicity, however, we can exploit the fact that this fluent is already present in the domain axiomatizations of ADL planning problems. In this way, we obtain the following translation into the Event Calculus.

1. The sorts, actions, and fluents are the same.
2. The foundational axioms on the time structure are augmented by (25).
3. The precondition axioms (19) are re-written to

$$Happens(A(\vec{x}), s) \supset \pi_A[s] \tag{26}$$

4. The set of effect axioms (20) is translated into the formulas

$$\begin{aligned} \Gamma_A^+ &\supset Initiates(A(\vec{x}), f, s) \\ \Gamma_A^- &\supset Terminates(A(\vec{x}), f, s) \end{aligned} \tag{27}$$

Predicates *Initiates* and *Terminates* are circumscribed locally, which results in the second-order axiom

$$CIRC\,[\,\bigvee_A(27); \mathit{Initiates}, \mathit{Terminates}\,] \tag{28}$$

5. The initial formula $I[0]$ is mapped onto a formula where each $\mathit{Holds}(f,0)$ is replaced by $\mathit{Init}_P(f)$ and each $\neg\mathit{Holds}(f,0)$ by $\mathit{Init}_N(f)$.
6. The encoding of a plan, (22), remains unchanged.

Example 4. (Continued) Recall the sub-formulas (23) of the general effect axiom (20). Assuming that `Move` is the only action in this domain with an actual effect, the corresponding Event Calculus definition of initiation and termination is equivalent to the following.

$$\begin{array}{l}\mathit{Initiates}(\texttt{Move}(\texttt{object},\texttt{old},\texttt{new}),\texttt{On}(\texttt{object},\texttt{new}),s)\\ \mathit{Initiates}(\texttt{Move}(\texttt{object},\texttt{old},\texttt{new}),\texttt{Above}(\texttt{object},\texttt{new}),s)\\ \mathit{Initiates}(\texttt{Move}(\texttt{object},\texttt{old},\texttt{new}),\texttt{Above}(\texttt{object},z),s)\ \subset\ \mathit{Holds}(\texttt{Above}(\texttt{new},z),s)\\ \\ \mathit{Terminates}(\texttt{Move}(\texttt{object},\texttt{old},\texttt{new}),\texttt{On}(\texttt{object},\texttt{old}),s)\\ \mathit{Terminates}(\texttt{Move}(\texttt{object},\texttt{old},\texttt{new}),\texttt{Above}(\texttt{object},z),s)\ \subset\ \mathit{Holds}(\texttt{Above}(\texttt{object},z),s)\end{array}$$

□

The equivalence of the domain constraints and precondition axioms are obvious. The correctness of the translation of the effect axioms is given by the following result.

Proposition 10. *Let Σ be a domain axiomatization resulting from an ADL planning problem, and let $\alpha_1,\ldots,\alpha_n$ be an action sequence such that $\Sigma \models \mathit{Poss}(\alpha_1,0,1)\wedge\ldots\wedge\mathit{Poss}(\alpha_n,n-1,n)$. Let Σ_{EC} be the mapping of Σ into the Event Calculus, then for any fluent ϕ*

$$\Sigma \models (\neg)\mathit{Holds}(\phi,t) \quad \textit{iff} \quad \Sigma_{EC} \models (\neg)\mathit{Holds}(\phi,t)$$

for all $t=1,\ldots,n$.

Proof: Suppose $\Sigma \models \mathit{Holds}(\phi,t)$, then by (20), $\Sigma \models \Gamma^+_{\alpha_t} \vee \mathit{Holds}(\phi,t-1)\wedge\neg\Gamma^-_{\alpha_t}$. Correspondingly, foundational axioms (25) imply that if $\Sigma_{EC} \models \mathit{Holds}(\phi,t)$, then Σ_{EC} entails either

1. $\mathit{Initiates}(\alpha_t,\phi,t-1)$, or
2. $(\exists m,s)\,(\mathit{Initiates}(\alpha_m,\phi,m-1)\wedge m<n-1\wedge\neg\mathit{Clipped}(m,\phi,n))$, or
3. $\mathit{Init}_P(\phi)\wedge\neg\mathit{Clipped}(0,\phi,n)$.

By (28), the first case is equivalent to $\Gamma^+_{\alpha_t}$ while the other two, by (25) and (28), are equivalent to $\mathit{Holds}(\phi,t-1)\wedge\neg\Gamma^-_{\alpha_t}$.

Suppose $\Sigma \models \neg\mathit{Holds}(\phi,t)$, then by (20), $\Sigma \models \Gamma^-_{\alpha_t} \vee \neg\mathit{Holds}(\phi,t-1)\wedge\neg\Gamma^+_{\alpha_t}$. Correspondingly, foundational axioms (25) imply that if $\Sigma_{EC} \models \neg\mathit{Holds}(\phi,t)$, then Σ_{EC} entails either

1. $\mathit{Terminates}(\alpha_t,\phi,t-1)$, or
2. $(\exists m,s)\,(\mathit{Terminates}(\alpha_m,\phi,m-1)\wedge m<n-1\wedge\neg\mathit{Declipped}(m,\phi,n))$, or
3. $\mathit{Init}_N(\phi)\wedge\neg\mathit{Declipped}(0,\phi,n)$.

By (28), the first case is equivalent to $\Gamma^-_{\alpha_t}$ while the other two, by (25) and (28), are equivalent to $\neg\mathit{Holds}(\phi,t-1)\wedge\neg\Gamma^+_{\alpha_t}$. ■

Together with Proposition 9 this shows that we have obtained a correct translation of ADL planning problems into the Event Calculus.

3.2. Translating the Fluent Calculus into the Situation Calculus

As a second result, we present a translation from the simple Fluent Calculus via the unifying calculus into the Situation Calculus based on Reiter's solution to the Frame Problem. As a by-product we obtain an extension of the latter suitable for nondeterministic actions.

3.2.1. From the Fluent Calculus ...

The Fluent Calculus is a variant of the Situation Calculus which uses the same branching time structure (cf. Example 2) and which adds to it a sort STATE as an explicit representation for states. Intuitively, a state is identified with the fluents that hold in it. The state in situation s is denoted by the standard function $State(s)$. By definition, each fluent itself is a (singleton) state, and if z_1 and z_2 are states, then so is their composition denoted by $z_1 \circ z_2$. The empty state is represented by the special constant $\emptyset$. The behavior of the function "$\circ$" is governed by the following foundational axioms, which essentially define states as non-nested sets of fluents. In the following, $Holds(f, z)$ is used as an abbreviation for the equational formula $(\exists z')\, z = f \circ z'$, which amounts to an axiomatic characterization of set membership.[4]

$$\begin{array}{ll} (z_1 \circ z_2) \circ z_3 = z_1 \circ (z_2 \circ z_3) & z_1 \circ z_2 = z_2 \circ z_1 \\ \neg Holds(f, \emptyset) & Holds(f_1, f) \supset f_1 = f \\ Holds(f, z_1 \circ z_2) \supset Holds(f, z_1) \vee Holds(f, z_2) & (\forall f)\,(Holds(f, z_1) \equiv Holds(f, z_2)) \supset z_1 = z_2 \\ (\forall P)(\exists z)(\forall f)\,(Holds(f, z) \equiv P(f)) & \end{array} \tag{29}$$

The last axiom, where P is a second-order predicate variable of sort FLUENT, stipulates the existence of a state for any (possibly infinite) set of fluents. These axioms are accompanied by the foundational axioms on situations, inherited from the Situation Calculus of [30]:

$$\begin{array}{l} (\forall s) \neg s < S_0 \\ (\forall a, a', s, s')\,(Do(a, s) = Do(a', s') \supset a = a' \wedge s = s') \\ (\forall a, s, s')\,(s < Do(a, s') \equiv s \leq s') \\ (\forall P)\,(P(S_0) \wedge (\forall a, s)\,(P(s) \supset P(Do(a, s))) \supset (\forall s)\, P(s)) \end{array} \tag{30}$$

The last axiom, where P is a second-order predicate variable of sort SIT, defines an induction principle over situations: if the initial situation satisfies a property P and this property is preserved through the execution of actions, then P is true for all situations.

Effects of actions are specified in the Fluent Calculus with the help of a purely axiomatic characterization of subtraction and addition of fluents from and to states:

$$\begin{array}{l} z_2 = z_1 + f \stackrel{\text{def}}{=} z_2 = z_1 \circ f \\ z_2 = z_1 - f \stackrel{\text{def}}{=} (z_2 = z_1 \vee z_2 \circ f = z_1) \wedge \neg Holds(f, z_2) \end{array}$$

These macros can be straightforwardly generalized to the subtraction and addition of finitely many fluents. On this basis, domains are axiomatized in the simple Fluent Calculus as follows, where the expression $Holds(f, s)$ in uniform state formulas (in the sense of Definition 2) stands for $Holds(f, State(s))$.

Definition 11. A *simple Fluent Calculus domain* is composed of the following elements.

1. Domain constraints, which are of the form
$$\delta[s] \tag{31}$$
where δ is a state formula in s.
2. Precondition axioms, one for every action $A(\vec{x})$, which are of the form
$$Poss(A(\vec{x}), s) \equiv \pi_A[s] \tag{32}$$
where $\pi_A[s]$ is a state formula in s with free variables among $s, \vec{x}$.
3. So-called state update axioms, one for every action $A(\vec{x})$, which are of the form
$$\begin{array}{ll} Poss(A(\vec{x}), s) \supset & (\exists \vec{y}_1)\,(\Phi_1[s] \wedge State(Do(A(\vec{x}), s)) = State(s) - \vartheta_1^- + \vartheta_1^+) \\ & \vee \ldots \vee \\ & (\exists \vec{y}_n)\,(\Phi_n[s] \wedge State(Do(A(\vec{x}), s)) = State(s) - \vartheta_n^- + \vartheta_n^+) \end{array} \tag{33}$$

[4] Below, the letter z always denotes variables of sort STATE.

where each $\Phi_i[s]$ is a state formula in s with free variables among $s, \vec{x}, \vec{y}_i$ and ϑ_i^- (the negative effects) and ϑ_i^+ (the positive effects) stand for zero or more subtractions and additions, respectively, of fluent terms with variables among $\vec{x}, \vec{y}_i$.

□

The basic theorem of the Fluent Calculus (see, e.g., [42]) says that the equations in state update axioms provide a solution to the Frame Problem.

Theorem 12. *Foundational axioms (29) entail that*

$$State(s') = State(s) - g_1 - \ldots - g_m + f_1 + \ldots + f_n$$

implies

$$Holds(f, State(s')) \equiv \bigvee_i f = f_i \vee \left[Holds(f, State(s)) \wedge \bigwedge_j f \neq g_j \right]$$

and vice versa.

A state update axiom (33) specifies an action with indeterminate effects if $n > 1$ and the conditions Φ_i are not mutually exclusive. But an action can also be nondeterministic if its state update axiom has a single update equation which is accompanied by an underspecified condition.

Example 5. Consider the following axiom, which specifies alternative forms of payments.

$$\begin{array}{l} Poss(\mathtt{Pay}, s) \supset \\ \quad (\exists y)\,(Holds(\mathtt{HasPayment}(y), s) \wedge State(Do(\mathtt{Pay}, s)) = State(s) - \mathtt{HasPayment}(y)) \end{array} \tag{34}$$

Suppose, for example, $Holds(\mathtt{HasPayment}(y), S_0) \equiv y = \mathtt{Cash} \vee y = \mathtt{Cheque}$, then according to Theorem 12 this state update axiom implies

$$\neg Holds(\mathtt{HasPayment}(\mathtt{Cash}), Do(\mathtt{Pay}, S_0)) \vee \neg Holds(\mathtt{HasPayment}(\mathtt{Cheque}), Do(\mathtt{Pay}, S_0))$$

but neither of the disjuncts alone is entailed. □

Based on Theorem 12, the translation of basic Fluent Calculus theories into our unifying calculus is straightforward. Domain constraints are taken as they are. A precondition axiom (32) is re-written as

$$Poss(A(\vec{x}), s, t) \equiv \pi_A[s] \wedge t = Do(A(\vec{x}), s) \tag{35}$$

A state update axiom (33) is mapped onto the effect axiom

$$\begin{array}{l} Poss(A(\vec{x}), s, t) \supset (\exists \vec{y}_1)(\Phi_1[s] \wedge (\forall f)\,[\bigvee_i f = f_{1i} \vee Holds(f, s) \wedge \bigwedge_j f \neq g_{1j} \supset Holds(f, t)] \\ \qquad\qquad\qquad\qquad \wedge (\forall f)\,[\bigvee_j f = g_{1j} \vee \neg Holds(f, s) \wedge \bigwedge_i f \neq f_{1i} \supset \neg Holds(f, t)]) \\ \qquad \vee \ldots \vee \\ \qquad (\exists \vec{y}_n)(\Phi_n[s] \wedge (\forall f)\,[\bigvee_i f = f_{ni} \vee Holds(f, s) \wedge \bigwedge_j f \neq g_{nj} \supset Holds(f, t)] \\ \qquad\qquad\qquad\qquad \wedge (\forall f)\,[\bigvee_j f = g_{nj} \vee \neg Holds(f, s) \wedge \bigwedge_i f \neq f_{ni} \supset \neg Holds(f, t)]) \end{array} \tag{36}$$

Here, the f_{ki} and g_{kj} are the fluent terms that occur in ϑ_k^+ and ϑ_k^-, respectively. The equivalence of this mapping is obvious for both domain constraints and precondition axioms. As the following proposition shows, correctness of the effect axioms follows if the updates are consistent, that is,

$$\bigwedge_i \bigwedge_j f_{ki} \neq g_{kj} \tag{37}$$

for all $k = 1, \ldots, n$.

Proposition 13. *Suppose* $Poss(A(\vec{x}), s)$ *and* $Poss(A(\vec{x}), s, t) \equiv t = Do(A(\vec{x}), s)$*, and assume that (37) holds for a state update axiom (33), then the foundational axioms of the Fluent Calculus imply that (33) and (36) are equivalent.*

Proof: Under the consistency assumption, the implication

$$\bigvee_j f = g_{kj} \vee [\neg Holds(f, s) \wedge \bigwedge_i f \neq f_{ki}] \supset \neg Holds(f, t)$$

is logically equivalent to

$$Holds(f, t) \supset [\bigwedge_j f \neq g_{kj} \wedge Holds(f, s)] \vee \bigvee_i f = f_{ki}$$

Hence, (36) can be equivalently written as

$$Poss(A(\vec{x}), s, t) \supset \bigvee_k (\exists \vec{y}_k)(\Phi_k[s] \wedge (\forall f)[\bigvee_i f = f_{ki} \vee Holds(f, s) \wedge \bigwedge_j f \neq g_{kj} \equiv Holds(f, t)]) \tag{38}$$

The equivalence of this effect axiom and state update axiom (33) follows immediately from Theorem 12. ■

Example 5. (Continued) Recall state update axiom (34) for the `Pay` action. The corresponding effect axiom is

$$\begin{array}{l} Poss(\texttt{Pay}, s, t) \supset \\ \quad (\exists y)\,(Holds(\texttt{HasPayment}(y), s) \wedge (\forall f)\,[Holds(f, s) \wedge f \neq \texttt{HasPayment}(y) \supset Holds(f, t)] \\ \qquad\qquad\qquad\qquad \wedge (\forall f)\,[\neg Holds(f, s) \vee f = \texttt{HasPayment}(y) \supset \neg Holds(f, t)]) \end{array}$$

which can be equivalently written as

$$\begin{array}{l} Poss(\texttt{Pay}, s, t) \supset \\ \quad (\exists y)\,(Holds(\texttt{HasPayment}(y), s) \wedge (\forall f)\,[Holds(f, s) \wedge f \neq \texttt{HasPayment}(y) \equiv Holds(f, t)]) \end{array} \tag{39}$$

□

3.2.2. . . . to the Situation Calculus

In the following, we show how a domain axiomatization resulting from a Fluent Calculus domain can be mapped onto the Situation Calculus using so-called successor state axioms [31] for each fluent $F(\vec{u})$ as a solution to the Frame Problem. The general form of these axioms in the reified version (that is, where fluents are represented as terms) is

$$Poss(a, s) \supset [Holds(F(\vec{u}), Do(a, s)) \equiv \Gamma_F^+[s] \vee Holds(F(\vec{u}), s) \wedge \neg\Gamma_F^-[s]] \tag{40}$$

Here, Γ_F^+ and Γ_F^- describe the conditions on $a, s, \vec{u}$ under which fluent $F(\vec{u})$ is, respectively, a positive or a negative effect. Reiter's basic action theories do not, however, allow to axiomatize nondeterministic actions. In [20], it has been shown how a generic predicate $Case(k, a, s)$, where $k : \mathbb{N}$, can be used in the sub-formulas Γ_F^+ and Γ_F^- to model nondeterministic effects by distinguishing different "cases" of updates. For a correct mapping of effect axioms of the form (38), however, this concept needs to be generalized in view of actions which are characterized by an underspecified condition, like action `Pay` (cf. axiom (34)). To this end, we introduce, for every action $A(\vec{x})$ with effect axiom (36), the more general predicates

$$Case_k^A(\vec{x}, \vec{y}_k, s)$$

for every $k = 1, \ldots, n$. The behavior of these predicates is governed by the following axioms.[5]

$$\begin{array}{l} Poss(A(\vec{x}), s) \supset \bigoplus_k (\exists \vec{y}_k)\,(Case_k^A(\vec{x}, \vec{y}_k, s) \wedge \Phi_k[s]) \\ (\forall \vec{y}_k, \vec{y}_k')\,(Case_k^A(\vec{x}, \vec{y}_k, s) \wedge Case_k^A(\vec{x}, \vec{y}_k', s) \supset \vec{y}_k = \vec{y}_k') \end{array} \tag{41}$$

[5] Below, the notation $\bigoplus_k F_k$ means that exactly one of the sub-formulas F_k is true.

Put in words, for every situation s in which the action is possible, there exists a unique applicable "case" k with a unique instance $\vec{y}_k$.

We are now in a position to map an axiomatization characterizing a Fluent Calculus domain into the Situation Calculus, provided the original Fluent Calculus domain satisfies the consistency assumption (37). Domain constraints are taken as they are while precondition axioms (35) are re-written to

$$Poss(A(\vec{x}), s) \equiv \pi_A[s] \tag{42}$$

Given in their equivalent form (38), the effect axioms are all together mapped onto the following schema for the (possibly nondeterministic) successor state axioms.

$$\begin{array}{l} Poss(a, s) \supset \\ \quad [Holds(f, Do(a, s)) \equiv \bigvee_A \; (\exists \vec{x})\,(a = A(\vec{x}) \wedge \bigvee_k (\exists \vec{y}_k)\,(Case_k^A(\vec{x}, \vec{y}_k, s) \wedge \bigvee_i f = f_{ki})) \\ \qquad\qquad\qquad\qquad \vee \\ \qquad\qquad\qquad\qquad (\exists \vec{x})\,(a = A(\vec{x}) \wedge Holds(f, s) \wedge \\ \qquad\qquad\qquad\qquad\qquad\qquad \neg[\bigvee_k (\exists \vec{y}_k)\,(Case_k^A(\vec{x}, \vec{y}_k, s) \wedge \bigvee_j f = g_{kj})])] \end{array} \tag{43}$$

The fluent variable f in this schema can then be instantiated by all fluents of the domain in order to obtain actual successor state axioms.

Example 5. (Continued) Recall general effect axiom (34) for the `Pay` action. Assuming this to be the only action in the domain, we obtain the following successor state axiom schema.

$$\begin{array}{l} Poss(a, s) \supset \\ \quad [Holds(f, Do(a, s)) \equiv a = \texttt{Pay} \wedge Holds(f, s) \wedge \neg[(\exists y)\, Case_1^{\texttt{Pay}}(y, s) \wedge f = \texttt{HasPayment}(y)]] \end{array} \tag{44}$$

along with the "case" axiom[6]

$$Poss(\texttt{Pay}, s) \supset (\exists! y)\,(Case_1^{\texttt{Pay}}(y, s) \wedge \texttt{HasPayment}(y, s))$$

Instantiating (44) for fluent $\{f \mapsto \texttt{HasPayment}(y)\}$, we obtain

$$\begin{array}{l} Poss(a, s) \supset \\ \quad [Holds(\texttt{HasPayment}(y), Do(a, s)) \equiv a = \texttt{Pay} \wedge Holds(\texttt{HasPayment}(y), s) \wedge \neg Case_1^{\texttt{Pay}}(y, s)] \end{array}$$

□

It is easy to prove that the mapping of the effect axioms into successor state axioms is correct.

Proposition 14. *Suppose* $Poss(A(\vec{x}), s)$ *and* $Poss(A(\vec{x}), s, t) \equiv t = Do(A(\vec{x}), s)$*, then (36) and (43), instantiated by* $\{a \mapsto A(\vec{x})\}$*, are equivalent under axioms (41).*

Proof: Instantiating (43) by $\{a \mapsto A(\vec{x})\}$, we obtain

$$\begin{array}{rl} Holds(f, Do(A(\vec{x}), s)) \equiv & \bigvee_k (\exists \vec{y}_k)\,(Case_k^A(\vec{x}, \vec{y}_k, s) \wedge \bigvee_i f = f_{ki}) \\ & \vee \\ & Holds(f, s) \wedge \neg[\bigvee_k (\exists \vec{y}_k)\, Case_k^A(\vec{x}, \vec{y}_k, s) \wedge \bigvee_j f = g_{kj}] \end{array}$$

Given (41), this can be re-written to

$$\bigvee_k (\exists \vec{y}_k)\,(\Phi_k[s] \wedge (\forall f)\,[Holds(f, Do(A(\vec{x}), s)) \equiv \bigvee_i f = f_{ki} \vee Holds(f, s) \wedge \neg[\bigvee_j f = g_{kj}]])$$

which, given $Poss(A(\vec{x}), s, t) \equiv t = Do(A(\vec{x}), s)$, is equivalent to (38). ■

We have thus obtained, with the help of the unifying calculus, an embedding of the full basic Fluent Calculus into the Situation Calculus using a variant of successor state axioms which is suitable for nondeterministic actions and somewhat more general than [20]. The translation itself generalizes an earlier result that was restricted to deterministic actions [35].

[6] Below, the notation $(\exists! y)F$ means the existence of a unique instance y such that sub-formula F is true.

3.3. Translations into Different Time Structures

The two example translations presented above have in common that the target language uses the same time structure as the input language. Since the unifying action calculus is not confined to a particular time structure, it can also serve as intermediary language for translating approaches with different time structures into each other. Domain axiomatizations with branching time, such as situations in the Situation- or the Fluent Calculus, can be mapped onto a linear time structure by introducing a special fluent to denote the actual occurrence of an action, like `Occurs`, and adding this to the precondition so that the possibility of an action can be identified with its actual occurrence. Conversely, domain axiomatizations with linear time (with time points t) can be mapped onto a branching time structure (with time points s) by adding a special fluent $Time(s,t)$ to denote that t is the "actual" time of a branching time point s. Precondition and effect axioms in the target language then inherit the relation between the beginning and end of an action as specified in the domain axioms with the linear time structure.

4. Modularity of Domain Axiomatizations

In this second part of the paper, we show how the unifying action calculus allows us to analyze and solve problems of general interest across different formalisms. The motivation for using the unifying approach is that it enables proofs of results without being confined to a specific approach. Once established, instantiating such a result to a particular action calculus is likely to be much easier than solving the problem individually and from the scratch for each individual formalism. We exemplify this by providing a new, calculus-independent solution to a problem that arises across different approaches: the question whether a domain axiomatization is modular. This problem, which has recently gained interest [15], is of particular relevance for the practical use of domain axiomatizations in high-level action programming languages like GOLOG [32] or FLUX [42], and it is intimately related to McCarthy's concept of elaboration tolerance [24].

The problem of modularity arises from the fact that axiomatizations of action domains combine different categories of formulas which serve different purposes. *Domain constraints* describe static properties which hold in all states; *precondition axioms* define the conditions for actions to be applicable; and *effect axioms* define the consequences of actions. As a uniform logical theory, however, a domain axiomatization may easily give rise to dependencies among the different kinds of axioms: effect formulas can entail implicit preconditions, domain constraints can entail implicit effects, etc. [13]. Implementations like GOLOG or FLUX, on the other hand, rely on the assumption that dependencies like these do not exist. The reason is that, for the sake of efficiency, the implementations use domain axiomatizations in a modular fashion: agents refer to the domain constraints only when they initialize their world model, they check the applicability of an action merely against the precondition axioms, and they update their world model entirely on the basis of the effect axioms. Agent programs would be much less efficient if the entire domain theory had to be taken into account for each specific reasoning task. However, this modular use of a domain axiomatization is incorrect whenever there is a dependency between axioms of different kind. As a consequence, the modularity of a domain axiomatization must always be verified prior to using it as the knowledge base for an agent. This is an excellent example of the value of McCarthy's elaboration tolerance principle [24]: the more dependencies there are between different types of axioms, the less elaboration tolerant is a formalism, because the addition of new information may disrupt an entire existing axiomatization if it is not modular [14].

In the following, we use our unifying action calculus as the formal basis for a general, calculus-independent analysis of the problem of modularity of domain constraints, precondition axioms, and effect formulas in domain axiomatizations. We present conditions for modularity against which a domain axiomatization can be checked. As the main result, we prove that the class of sequential and ramification-free domain axiomatizations, as defined at the end of Section 2, are guaranteed to be free of dependencies if they satisfy these conditions. We then show how this result can be straightforwardly instantiated to several concrete formalisms. In this way, our general conditions for independence can be easily checked, e.g., by applying automated theorem proving, for a given domain theory in a specific action calculus.

4.1. Examples for Implicit Dependency

In order to illustrate the universality of the problem of modularity, we first present three simple example axiomatizations in three different calculi which entail implicit domain constraints, preconditions, and effects, respectively.

4.1.1. Implicit Domain Constraints

The first example, axiomatized in the Situation Calculus, shows how effect axioms—successor state axioms in this case—may entail additional, implicit domain constraints.

Example 6. For a scheduling domain consider the fluents $\texttt{Job}(m, j)$ and $\texttt{Free}(m)$, respectively representing that machine m has been allocated job j and that machine m is free. Two actions $\texttt{Schedule}(j, m)$ and $\texttt{Unschedule}(j)$ are for allocating job j to machine m and for deallocating job j.

Consider the single domain constraint[7]

$$\mathit{Holds}(\texttt{Free}(m), t) \supset \neg(\exists j)\, \mathit{Holds}(\texttt{Job}(m, j), t) \tag{45}$$

Let the action precondition axioms be

$$\begin{array}{l} \mathit{Poss}(\texttt{Schedule}(j, m), s) \equiv \mathit{Holds}(\texttt{Free}(m), s) \\ \mathit{Poss}(\texttt{Unschedule}(j), s) \equiv (\exists m)\, \mathit{Holds}(\texttt{Job}(m, j), s) \end{array} \tag{46}$$

The successor state axioms for the two fluents are as follows.

$$\begin{array}{l} \mathit{Poss}(a, s) \supset \\ \quad [\, \mathit{Holds}(\texttt{Job}(m, j), \mathit{Do}(a, s)) \equiv a = \texttt{Schedule}(j, m) \vee \\ \qquad\qquad\qquad\qquad \mathit{Holds}(\texttt{Job}(m, j), s) \wedge a \neq \texttt{Unschedule}(j)\,] \\ \mathit{Poss}(a, s) \supset \\ \quad [\, \mathit{Holds}(\texttt{Free}(m), \mathit{Do}(a, s)) \equiv (\exists j)\, (\mathit{Holds}(\texttt{Job}(m, j), s) \wedge a = \texttt{Unschedule}(j)) \vee \\ \qquad\qquad\qquad\qquad \mathit{Holds}(\texttt{Free}(m), s) \wedge \neg(\exists j)\, a = \texttt{Schedule}(j, m)\,] \end{array} \tag{47}$$

Put in words, $\texttt{Job}(m, j)$ holds in a successor situation if job j was just allocated to machine m, or if $\texttt{Job}(m, j)$ was true beforehand and j was not deallocated. Similarly, $\texttt{Free}(m)$ holds in a successor situation if the job allocated to m just got unscheduled, or if machine m was free beforehand and has not just got some job j.

We claim that this axiomatization entails the following implicit domain constraint (which does not follow from (45) alone):

$$\mathit{Holds}(\texttt{Job}(m, j), s) \supset \neg(\exists i)\, (\mathit{Holds}(\texttt{Job}(m, i), s) \wedge j \neq i) \tag{48}$$

To see why, suppose $\mathit{Holds}(\texttt{Job}(m, j), s)$, then (46) implies $\mathit{Poss}(\texttt{Unschedule}(j), s)$. By (47),

$$\mathit{Holds}(\texttt{Free}(m), \mathit{Do}(\texttt{Unschedule}(j), s))$$

Hence, from (45) it follows that

$$\neg(\exists i)\, \mathit{Holds}(\texttt{Job}(m, i), \mathit{Do}(\texttt{Unschedule}(j), s)) \tag{49}$$

Also, by (47) and uniqueness-of-names,

$$\mathit{Holds}(\texttt{Job}(m, i), \mathit{Do}(\texttt{Unschedule}(j), s)) \equiv \mathit{Holds}(\texttt{Job}(m, i), s) \wedge j \neq i$$

This and (49) imply $\neg(\exists i)\, (\mathit{Holds}(\texttt{Job}(m, i), s) \wedge j \neq i)$. □

[7] It should be stressed that the converse of the following implication is left out intentionally; that is to say, for many other reasons a machine may not be available even if it has not been allocated a job.

4.1.2. Implicit Preconditions

The next example, which is axiomatized in the simple Event Calculus as introduced in Section 3.1.2, shows how effect axioms can give rise to implicit preconditions of actions.

Example 7. To model the movement of a robot, consider the fluent $\mathtt{At}(x)$ representing that the robot is at position x. The action $\mathtt{Go}(x, y)$ denotes the movement of the robot from x to y. Let the domain axiomatization consist of the single domain constraint

$$\mathit{Holds}(\mathtt{At}(l_1), t) \wedge \mathit{Holds}(\mathtt{At}(l_2), t) \supset l_1 = l_2 \tag{50}$$

Put in words, the robot must be at a unique location at any time. Let the precondition of the only action be axiomatized as

$$\mathit{Happens}(\mathtt{Go}(x, y), s) \supset \mathit{Holds}(\mathtt{At}(x), s) \tag{51}$$

The effects in this domain are determined by the following circumscribed definition of initiation and termination.

$$\begin{aligned} \mathit{Initiates}(a, f, s) &\equiv (\exists x, y)\,(f = \mathtt{At}(y) \wedge a = \mathtt{Go}(x, y)) \\ \mathit{Terminates}(a, f, s) &\equiv (\exists x, y)\,(f = \mathtt{At}(x) \wedge a = \mathtt{Go}(x, y)) \end{aligned} \tag{52}$$

We claim that, under the assumption that the $\mathtt{Go}$ action can be performed in isolation, this axiomatization entails the following implicit precondition (which does not follow from (51) alone):

$$\mathit{Happens}(\mathtt{Go}(x, y), s) \supset x \neq y \tag{53}$$

To see why, (52) implies both $\mathit{Initiates}(\mathtt{Go}(x, y), \mathtt{At}(y), s)$ and $\mathit{Terminates}(\mathtt{Go}(x, y), \mathtt{At}(x), s)$. Suppose $\mathit{Happens}(a, s) \equiv a = \mathtt{Go}(x, y)$, then foundational axioms (25) entail both $\mathit{Holds}(\mathtt{At}(y), t)$ as well as the negation $\neg\mathit{Holds}(\mathtt{At}(x), t)$ for $t > s$, which in turn implies $x \neq y$. □

4.1.3. Implicit Effects

The last example, given in the basic Fluent Calculus, shows how domain constraints can give rise to additional, implicit effects.

Example 8. To model the operation of two elevators, consider the fluent $\mathtt{AtFloor}(e, n)$ with $e \in \{\mathtt{E}_1, \mathtt{E}_2\}$ and $n \in \{0, 1, \ldots, 9\}$, representing the current floor of each elevator. The only action $\mathtt{Call}(n)$ means to activate the call button at floor n. Let the domain axiomatization consist of the two domain constraints,

$$\begin{aligned} &(\exists!k)\ \mathit{Holds}(\mathtt{AtFloor}(e, k), t) \\ &\neg\mathit{Holds}(\mathtt{AtFloor}(\mathtt{E}_1, 0), t) \end{aligned} \tag{54}$$

that is to say, both elevators are at a unique floor in every situation, and the first elevator does not serve the basement. We assume that it is possible to activate the call button at any floor as long as there is no elevator at this floor, that is,

$$\mathit{Poss}(\mathtt{Call}(n), s) \equiv 0 \leq n \leq 9 \wedge \neg(\exists e)\,\mathit{Holds}(\mathtt{AtFloor}(e, n), s) \tag{55}$$

The following state update axiom specifies a nondeterministic effect.

$$\begin{aligned} \mathit{Poss}(\mathtt{Call}(n), s) \supset\ & ((\exists m)\,(\,\mathit{Holds}(\mathtt{AtFloor}(\mathtt{E}_1, m), s) \wedge \\ & \qquad \mathit{State}(\mathit{Do}(\mathtt{Call}(n), s)) = \mathit{State}(s) - \mathtt{AtFloor}(\mathtt{E}_1, m) + \mathtt{AtFloor}(\mathtt{E}_1, n))) \\ & \vee \\ & ((\exists m)\,(\,\mathit{Holds}(\mathtt{AtFloor}(\mathtt{E}_2, m), s) \wedge \\ & \qquad \mathit{State}(\mathit{Do}(\mathtt{Call}(n), s)) = \mathit{State}(s) - \mathtt{AtFloor}(\mathtt{E}_2, m) + \mathtt{AtFloor}(\mathtt{E}_2, n))) \end{aligned} \tag{56}$$

Put in words, calling an elevator to a floor n has the indeterminate effect that either of the two elevators arrives.

We claim that this domain axiomatization entails the following implicit effect (which does not follow from (56) alone):

$$Poss(\texttt{Call}(0), s) \supset Holds(\texttt{AtFloor}(\texttt{E}_2, 0), Do(\texttt{Call}(0), s)) \tag{57}$$

To see why, note that $State(Do(\texttt{Call}(0), s)) = State(s) - \texttt{AtFloor}(\texttt{E}_1, m) + \texttt{AtFloor}(\texttt{E}_1, 0)$ implies

$$Holds(\texttt{AtFloor}(\texttt{E}_1, 0), Do(\texttt{Call}(0), s))$$

according to Theorem 12. By (54), therefore, the first disjunct in state update axiom (56) is false if substituted by $\{n \mapsto 0\}$. This entails (57) according to (56) and Theorem 12. □

4.2. A General Method for Verifying Modularity

The examples in the previous section show that the problem of domain axiomatizations not being modular arises in many different action formalisms. With the help of our unifying action calculus, we can give a general, formal definition of what are implicit domain constraints, preconditions, and effects. To this end, we introduce the following notation for a given action $A(\vec{x})$. In a domain axiomatization with precondition axioms Π, by Π_A we denote the one which is for $A(\vec{x})$, with $\pi_A[s]$ being its right-hand side as usual. Likewise, if Υ are the effect axioms, then by Υ_A we denote the one for action $A(\vec{x})$. For notational convenience, we will refrain from stating the foundational axioms of a domain axiomatization. These are tacitly assumed to be satisfiable, and entailment ($\models$) and consistency of sets of formulas is always meant to be modulo them.

Definition 15. Consider a domain axiomatization $\Sigma = \Delta \cup \Pi \cup \Upsilon$ consisting of domain constraints Δ, precondition axioms Π, and effect axioms Υ.

1. The domain axiomatization is *free of implicit domain constraints* if for every state formula $\delta[t]$,

   $$\Sigma \models \delta[t]$$

   implies $\Delta \models \delta[t]$.
2. The domain axiomatization is *free of implicit preconditions* if for every action $A(\vec{x})$ and state formula $\pi[s]$,

   $$\Sigma \models Poss(A(\vec{x}), s, t) \supset \pi[s]$$

   implies $\Delta \cup \Pi_A \models Poss(A(\vec{x}), s, t) \supset \pi[s]$.
3. The domain axiomatization is *free of implicit effects* if for every action $A(\vec{x})$ and state formula $\varepsilon[t]$,

   $$\Sigma \models Poss(A(\vec{x}), s, t) \supset \varepsilon[t]$$

   implies $\Delta[S] \cup \Pi_A[S] \cup \Upsilon_A[S, T] \models Poss(A(\vec{x}), S, T) \supset \varepsilon[T]$, for any constants S, T of sort TIME.

□

Put in words, an implicit domain constraint is a (universally quantified) state formula which is entailed by the entire domain axiomatization but which cannot be derived from the given domain constraints Δ alone. An implicit precondition is entailed by the entire domain axiomatization but does not follow from the precondition axioms alone in a state that satisfies the domain constraints. The rationale behind this definition is the following: given a state that satisfies the domain constraints Δ, the precondition axiom for an action A alone should suffice to entail all executability conditions for this action. Finally, an implicit effect follows from the entire domain axiomatization but not from an effect axiom alone in a state that satisfies the preconditions of an action and the domain constraints. The rationale behind this definition is this: given a state that satisfies both the domain constraints Δ and the preconditions of an action A, the instantiated effect axiom for this action alone should suffice to infer everything that can be concluded of the resulting state. The use of TIME constants in item 3 is motivated by the desire to verify modularity in a local fashion, that is, by instantiating the domain constraints and precondition axioms by a single time point, and the effect axioms by this time point and its successor.

We now use our unifying calculus to provide three conditions which will then be shown to guarantee that a domain axiomatization is free of implicit dependencies. Informally speaking, the first condition below, (C1), essentially says that for every state at some time S which is consistent with the domain constraints and in which an action $A(\vec{x})$ is applicable, the condition $\Phi_i[S]$ for at least one case i in the effect axiom for A holds. Condition (C2) implies that none of the applicable effect specifications is self-contradictory, and (C3) requires that any possible update leads to a state that satisfies the domain constraints. Here and in the following, we consider only ramification-free domain axiomatizations according to Definition 6, so that the sub-formulas Γ_i^+ and Γ_i^- in effect axioms (cf. (2)) are state formulas solely in s.

Definition 16. Let S, T be constants of sort TIME. A domain axiomatization $\Delta \cup \Pi \cup \Upsilon$ is called *modular* if the following holds for every action $A(\vec{x})$ with effect axiom (1): there exist arbitrary TIME constants S, T such that

$$\models \Delta[S] \wedge \pi_A[S] \supset \bigvee_{i=1}^{k} (\exists \vec{y}_i) \Phi_i[S] \tag{C1}$$

and, for all $i \in \{1, \ldots, k\}$,

$$\models \Delta[S] \wedge \pi_A[S] \wedge \Phi_i[S] \wedge \Gamma_i^+[S] \supset \neg \Gamma_i^-[S] \tag{C2}$$

$$\models \Delta[S] \wedge \pi_A[S] \wedge \Upsilon_i[S, T] \supset \Delta[T] \tag{C3}$$

□

These conditions can in principle be checked for a given domain axiomatization by automated theorem proving.[8] When so doing, an advantage is that these conditions can be verified separately for each action of a domain signature.

We are now ready to prove our main result, which says that modular domain axiomatizations are free of implicit domain constraints, preconditions, and effects. We begin by proving that if a state formula is consistent with the domain constraints, then it is also consistent with the entire domain axiomatization.

Lemma 17. *Consider a sequential and ramification-free domain axiomatization $\Delta \cup \Pi \cup \Upsilon$ which satisfies conditions (C1)–(C3). Let S be an arbitrary constant of sort* TIME *and $\psi(S)$ a state formula in S, then*

$$\Delta[S] \cup \{\psi[S]\} \quad \textit{is consistent} \tag{58}$$

implies that $\Delta \cup \Pi \cup \Upsilon \cup \{\psi[S]\}$ is consistent.

Proof: Let $\mathcal{I}'$ be an arbitrary model for $\Delta[S] \cup \{\psi[S]\}$. From this we can straightforwardly obtain a model $\mathcal{I}$ in which $S^{\mathcal{I}}$ is the least element of $<^{\mathcal{I}}$. We then construct a model $\mathcal{J}$ as follows.

1. For every fluent φ,
$$\mathit{Holds}(\varphi, S)^{\mathcal{J}} \quad \text{iff} \quad \mathit{Holds}(\varphi, S)^{\mathcal{I}} \tag{59}$$
2. Given that the domain axiomatization is sequential, we can iteratively construct the following assignment for every σ in the domain of $\mathcal{I}$ for sort TIME, starting with $S^{\mathcal{I}}$, and for every action $\alpha = A(\vec{x})^{\mathcal{I}}$ and time point $\tau >^{\mathcal{I}} \sigma$:
   (a) Let π_α be the right-hand side of the precondition axiom for A, then
$$\mathit{Poss}(\alpha, \sigma, \tau)^{\mathcal{J}} \quad \text{iff} \quad \mathcal{J} \models \pi_\alpha\{s \mapsto \sigma, t \mapsto \tau\} \tag{60}$$

[8] It is worth noting that condition (C2) is trivially true for both successor state axioms in the Situation Calculus and (consistent) state update axioms in the Fluent Calculus, because the corresponding formulas Γ_i^+ and Γ_i^- in the general effect axioms are always negations of each other (cf. axiom (38)).

(b) Let (1) be α's effect axiom. If $Poss(\alpha, \sigma, \tau)^{\mathcal{J}}$ then choose some $i = 1, \ldots, k$ and some $\vec{y}_i$ such that $\Phi_i[\sigma]^{\mathcal{J}}$, and for every fluent φ let

$$\begin{array}{r} Holds(\varphi, \tau)^{\mathcal{J}} \text{ if } \mathcal{J} \models \Gamma_i^+[\sigma]\{f \mapsto \varphi\} \\ \neg Holds(\varphi, \tau)^{\mathcal{J}} \text{ if } \mathcal{J} \models \Gamma_i^-[\sigma]\{f \mapsto \varphi\} \end{array} \tag{61}$$

The existence of some such i and $\vec{y}_i$ is guaranteed by assumption (C1), and consistency of the assignment (61) follows from assumption (C2).

Then $\mathcal{J}$ is a model for $\psi[S]$ due to (59), for Π due to (60), for Υ due to (61), and for Δ due to (58) and (C3). ∎

Next, we show that a state formula $\psi[T]$ is consistent with the entire domain axiomatization if only it is consistent with an instance of an update $\Upsilon_A[S, T]$ for a state at time S that satisfies the domain constraints and the preconditions of action A.

Lemma 18. *Consider a sequential and ramification-free domain axiomatization $\Sigma = \Delta \cup \Pi \cup \Upsilon$ which satisfies conditions (C1)–(C3). Let $A(\vec{x})$ be an action, S, T be arbitrary constants of sort* TIME*, and $\psi[T]$ a state formula in T, then*

$$\Delta[S] \cup \{\pi_A[S]\{t \mapsto T\}\} \cup \Upsilon_A[S, T] \cup \{\psi[T]\} \quad \textit{is consistent}$$

implies

$$\Sigma \cup \{Poss(A(\vec{x}), S, T)\} \cup \{\psi[T]\} \quad \textit{is consistent.}$$

Proof: Let $\mathcal{I}$ be a model for $\Delta[S] \cup \{\pi_A[S]\{t \mapsto T\}\} \cup \Upsilon_A[S, T] \cup \{\psi[T]\}$. We construct a model $\mathcal{J}$ as in the proof of Lemma 17 and with a specific assignment for the state at time T:

$$Holds(\varphi, T)^{\mathcal{J}} \text{ iff } Holds(\varphi, T)^{\mathcal{I}} \tag{62}$$

for every φ in the domain of $\mathcal{I}$ for sort FLUENT. This is consistent with Υ because $\mathcal{I}$ is a model for $\Upsilon_A[S, T]$. As above, $\mathcal{J}$ is a model for Σ, a model for $\psi[T]$ due to (62), and for $Poss(A(\vec{x}), S, T)$ since $\mathcal{I}$ is a model for $\pi_A[S]\{t \mapsto T\}$. ∎

With the help of these two lemmas we can now prove our main result.

Theorem 19. *Any sequential and ramification-free domain axiomatization which is modular is free of implicit domain constraints, preconditions, and effects.*

Proof: Let Σ be a modular domain axiomatization with domain constraints Δ, precondition axioms Π, and effect axioms Υ.

Consider an arbitrary state formula $\delta[t]$. If $\Sigma \models \delta[t]$ then $\Sigma \cup \{\neg\delta[t]\}$ is inconsistent. By Lemma 17, $\Delta \cup \{\neg\delta[t]\}$ is inconsistent, hence $\Delta \models \delta[t]$. This shows that Σ is free of implicit domain constraints.

Consider a state formula $\pi[s]$. If $\Sigma \models Poss(A(\vec{x}), s, t) \supset \pi[s]$ then $\Sigma \models \pi_A[s] \supset \pi[s]$, where π_A is the right-hand side of the precondition axiom for $A(\vec{x})$. Hence, $\Sigma \cup \{\pi_A[s] \wedge \neg\pi[s]\}$ is inconsistent. By Lemma 17, $\Delta \cup \{\pi_A[s] \wedge \neg\pi[s]\}$ is inconsistent, hence $\Delta \models \pi_A[s] \supset \pi[s]$, hence $\Delta \cup \Pi \models Poss(A(\vec{x}), s, t) \supset \pi[s]$. This shows that Σ is free of implicit preconditions.

Finally, consider an action $A(\vec{x})$ along with a state formula $\varepsilon[t]$. If $\Sigma \models Poss(A(\vec{x}), s, t) \supset \varepsilon[t]$ then $\Sigma \cup \{\pi_A[S]\{t \mapsto T\} \wedge \neg\varepsilon[T]\}$ is inconsistent for any S, T of sort TIME. By Lemma 18, $\Delta[S] \cup \{\pi_A[S]\{t \mapsto T\}\} \cup \Upsilon_A[S, T] \cup \{\neg\varepsilon[T]\}$ is inconsistent, hence $\Delta[S] \cup \Pi_A[S] \cup \Upsilon_A[S, T] \models Poss(A(\vec{x}), S, T) \supset \varepsilon[T]$. This shows that Σ is free of implicit effects. ∎

We conclude our analysis by illustrating how the general method can be easily instantiated and applied in order to verify independence in each of the specific approaches of the Situation-, Event-, and Fluent Calculus.

4.2.1. Modularity in the Situation Calculus

In Section 3.2.2 we have seen how a specific class of domain axiomatizations in the unifying action calculus can be mapped onto successor state axioms. The converse translation of action theories in the Situation Calculus consisting of domain constraints, precondition axioms, and basic successor state axioms in the sense of [31] is straightforward: domain constraints are taken as they are, precondition axioms of the form (42) are re-written into the form (35), and the successor state axioms (40) for all fluents F together are mapped onto the effect axiom schema

$$\begin{aligned} \mathit{Poss}(a,s,t) \supset\ & (\forall f)\,[\textstyle\bigvee_F(f = F(\vec{u}) \wedge (\Gamma^+_F[s] \vee \mathit{Holds}(f,s) \wedge \neg\Gamma^-_F[s]) \supset \mathit{Holds}(f,t))] \\ & \wedge \\ & (\forall f)\,[\textstyle\bigvee_F(f = F(\vec{u}) \wedge (\Gamma^-_F[s] \vee \neg\mathit{Holds}(f,s) \wedge \neg\Gamma^+_F[s]) \supset \neg\mathit{Holds}(f,t))] \end{aligned}$$

The action variable a in this schema can then be instantiated by all actions of the domain in order to obtain actual effect axioms in the unifying calculus. By definition, these axiomatizations are ramification-free, and the foundational axioms of the Situation Calculus, (30), imply sequentiality according to Definition 5. Based on this translation, the verification of the modularity conditions in Situation Calculus axiomatizations is straightforward.

Example 6. (Continued) We have seen that the given axiomatization entails an implicit domain constraint. Indeed, independence condition (C3) is not entailed. To see why, take arbitrary TIME constants S and T and consider the action $\texttt{Unschedule}(j)$. Successor state axioms (47) determine an update formula which is equivalent to

$$\begin{aligned} \Upsilon_1[S,T] := \ & [\mathit{Holds}(\texttt{Job}(m,i),T) \equiv \mathit{Holds}(\texttt{Job}(m,i),S) \wedge i \neq j] \wedge \\ & [\mathit{Holds}(\texttt{Free}(m),T) \equiv \mathit{Holds}(\texttt{Job}(m,j),S) \vee \mathit{Holds}(\texttt{Free}(m),S)] \end{aligned}$$

Along with

$$\begin{aligned} \Delta[S] &:= \mathit{Holds}(\texttt{Free}(n),S) \supset \neg(\exists k)\,\mathit{Holds}(\texttt{Job}(n,k),S) \\ \pi_{\texttt{Unschedule}}[S] &:= (\exists m)\,\mathit{Holds}(\texttt{Job}(m,j),S) \end{aligned}$$

this does *not* entail

$$\Delta[T] := \mathit{Holds}(\texttt{Free}(n),T) \supset \neg(\exists k)\,\mathit{Holds}(\texttt{Job}(n,k),T)$$

To see why, consider an interpretation that satisfies

$$\begin{aligned} & \mathit{Holds}(\texttt{Job}(M,J),S),\ \mathit{Holds}(\texttt{Job}(M,I),S),\ \neg\mathit{Holds}(\texttt{Free}(M),S) \\ & \neg\mathit{Holds}(\texttt{Job}(M,J),T),\ \mathit{Holds}(\texttt{Job}(M,I),T),\ \mathit{Holds}(\texttt{Free}(M),T) \end{aligned}$$

It is easy to verify that this is a model for the conjunction $\Delta[S] \wedge \pi_{\texttt{Unschedule}}[S] \wedge \Upsilon_1[S,T]\{i \mapsto I, j \mapsto J\}$ but not for $\Delta[T]$.[9] □

4.2.2. Modularity in the Event Calculus

In Section 3.1.2 we have seen how a specific class of domain axiomatizations in the unifying action calculus can be mapped onto the simple Event Calculus. This translation can be easily reversed. The only required generalization is to additionally axiomatize the special action `Inert` as in Example 3 to capture arbitrary sequential narratives, based on an arbitrary linear time structure (like, e.g., the positive real numbers) and where actions may not immediately follow each other. With regard to verifying modularity, it is easy to see that the axioms for `Inert` (cf. (12) and (13)) satisfy the conditions (C1)–(C3).

Example 7. (Continued) We have seen that the given axiomatization entails an implicit precondition. Indeed, independence condition (C2) is not entailed. To see why, take an arbitrary TIME constant S

[9] The reader may verify that (C3) *is* entailed, however, once the implicit domain constraint (48) is added.

and consider the action $\texttt{Go}(x, y)$. Initiation and termination axioms (52) determine effect formulas in the unifying action calculus where

$$\begin{aligned}\Gamma_1^+[S] &:= f = \texttt{At}(y) \vee \mathit{Holds}(f, S) \wedge f \neq \texttt{At}(x)\\ \Gamma_1^-[S] &:= f = \texttt{At}(x) \vee \neg\mathit{Holds}(f, S) \wedge f \neq \texttt{At}(y)\end{aligned}$$

Then $\Gamma_1^+[S]$ in conjunction with

$$\begin{aligned}\Delta[S] &:= \mathit{Holds}(\texttt{At}(l_1), S) \wedge \mathit{Holds}(\texttt{At}(l_2), S) \supset l_1 = l_2\\ \Phi_1[S] &:= \top\\ \pi_{\texttt{Go}}[S] &:= \mathit{Holds}(\texttt{At}(x), S)\end{aligned}$$

does *not* entail $\neg\Gamma_1^-[S]$. This can be easily seen by an interpretation that satisfies $x = y$.[10] □

4.2.3. Modularity in the Fluent Calculus

In Section 3.2.1 we have shown how basic Fluent Calculus theories can be mapped onto domain axiomatizations in the unifying calculus. This mapping can be directly applied to verify modularity of these theories with the help of our general method.

Example 8. (Continued) We have seen that the given axiomatization entails an implicit effect. Indeed, independence condition (C3) is not entailed. To see why, take arbitrary TIME constants S and T and consider the action $\texttt{Call}(n)$. State update axiom (56) determines an effect formula in the unifying action calculus where

$$\begin{aligned}\Upsilon_1[S,T] := (\exists m)\,(\,&\mathit{Holds}(\texttt{AtFloor}(\texttt{E}_1, m), S) \wedge\\ &(f = \texttt{AtFloor}(\texttt{E}_1, n) \vee \mathit{Holds}(f, S) \wedge f \neq \texttt{At}(\texttt{E}_1, m) \supset \mathit{Holds}(f, T)) \wedge\\ &(f = \texttt{At}(\texttt{E}_1, m) \vee \neg\mathit{Holds}(f, S) \wedge f \neq \texttt{AtFloor}(\texttt{E}_1, n) \supset \neg\mathit{Holds}(f, T))\,)\\ \Upsilon_2[S,T] := (\exists m)\,(\,&\mathit{Holds}(\texttt{AtFloor}(\texttt{E}_2, m), S) \wedge\\ &(f = \texttt{AtFloor}(\texttt{E}_2, n) \vee \mathit{Holds}(f, S) \wedge f \neq \texttt{At}(\texttt{E}_2, m) \supset \mathit{Holds}(f, T)) \wedge\\ &(f = \texttt{At}(\texttt{E}_2, m) \vee \neg\mathit{Holds}(f, S) \wedge f \neq \texttt{AtFloor}(\texttt{E}_2, m) \supset \neg\mathit{Holds}(f, T))\,)\end{aligned}$$

For the instance $\{n \mapsto 0\}$, $\Upsilon_1[S,T]$ along with

$$\begin{aligned}\Delta[S] &:= (\exists!k)\,\mathit{Holds}(\texttt{AtFloor}(e, k), S) \wedge \neg\mathit{Holds}(\texttt{AtFloor}(\texttt{E}_1, 0), S)\\ \pi_{\texttt{Call}}[S] &:= 0 \leq n \leq 9 \wedge \neg(\exists e)\,\mathit{Holds}(\texttt{AtFloor}(e, n), s)\end{aligned}$$

implies $\mathit{Holds}(\texttt{At}(\texttt{E}_1, 0), T)$. This, however, contradicts $\Delta[T]$.[11] □

5. Discussion

We have proposed a unifying action calculus which abstracts from a concrete time structure and a specific solution to the Frame Problem and thus encompasses a variety of existing, specific languages for axiomatizing action domains. This unifying approach can be used as an intermediary language to facilitate translations of specific calculi into each other. We have exemplified this by obtaining two new results: a characterization of ADL planning problems in the Event Calculus and an embedding of the full basic Fluent Calculus into a variant of the Situation Calculus with nondeterministic successor state axioms. Generally speaking, the use of an intermediary axiomatization has two major advantages. First, it makes explicit how the specific solution to the Frame Problem in the input language determines the effects and non-effects of an action. This often makes it easier to find an appropriate translation into a different solution to the Frame

[10] The reader may verify that (C2) *is* entailed, however, once the implicit precondition (53) is added.

[11] The reader may verify that (C3) *is* entailed, however, once the implicit effect (57) is incorporated into the state update axiom.

Problem. As an example, the domain axiomatization we have obtained in Section 3.2.1 from a basic Fluent Calculus theory not only gives a clear indication of how the effect axioms can be translated into successor state axioms, it also illustrates very explicitly what extension of Reiter's basic action theories is needed in order to capture the various ways in which nondeterministic actions can be axiomatized by state update axioms in the Fluent Calculus. Second, the prior translation of an input language into an intermediary language should allow for a generic and uniform way of embedding the input formalism into different target languages. The domain axiomatization we have obtained in Section 3.1.1 as a characterization for ADL planning problems, for example, can be readily used to define mappings into approaches other than the Event Calculus.

Among the variety of potential applications of inter-calculi translations, we consider the following ones most important.

1. Translations can be used to prove that (a well-defined class of) a specific calculus can be formally embedded in another calculus.
2. The use of an implementation of a calculus to solve problems given in a different input language requires a prior translation; examples are the problem specification languages used for the Planning Competitions [6] or the General Game Description Language used for the General Game Playing Contest [8].
3. In order to use a different platform to run knowledge-based agent programs written in languages like GOLOG or FLUX, the background knowledge of the agent needs to be transformed into an appropriate encoding.

In comparison to related work, much like the systematic assessment methods of [33] or the Action Description Language $\mathcal{A}$ [7] and extensions thereof, our unifying calculus can be used to analyze the relative expressiveness of different axiomatization techniques in comparison. The main difference is that the former define a specific semantics for action domains rather than providing a purely logical axiomatization. This implies that the assessments are always restricted to problem classes that can be defined within the special semantics. For example, Action Description Language $\mathcal{A}$ has been translated into both successor state axioms [16] and state update axioms [39]. These results can be combined into a translation from the Fluent Calculus into the Situation Calculus and vice versa, but this translation is confined to domains that can be expressed in $\mathcal{A}$ and therefore does not allow for a full embedding of basic Fluent Calculus theories into the Situation Calculus.

In the second part of the paper, we have used the unifying calculus to develop a general method for verifying independence of domain constraints, preconditions, and effects in axiomatizations of action domains. We have shown how this general method can be easily instantiated for various specific calculi. Existing results on the problem of implicit dependencies are restricted to specific calculi and less general classes of domains. In [30], it has been shown that precondition axioms and deterministic successor state axioms in the Situation Calculus are always independent, provided that there are no domain constraints at all. In [15], algorithms have been presented for inferring implicit domain constraints and preconditions from domain axiomatizations given in propositional modal logic. A conceptually different approach has been pursued in [27], where it has been shown how a particular class of domain constraints can be compiled into successor state axioms (deterministic only). Incidentally, condition (C3) in our Definition 16 is already known in this context as a way to ensure that, if satisfied by a given initial situation, the result of such a compilation allows to ignore the domain constraints. In our context, however, this condition serves a different purpose: instead of showing that, for specific initial situations, the *given* domain constraints are redundant, it shows that no *further* domain constraints are entailed (independent of the initial situation). With regard to the Event Calculus, it should be stressed that our result is restricted to sequential domains. However, our unifying calculus can be readily used to express concurrent actions, simply by taking as the elements of the sort ACTION collections of (simultaneous or overlapping) actions. It remains an issue for future work to define a concrete sound and complete mapping of domains with concurrent actions given, say, in the Event Calculus, into our unifying calculus in which single actions may represent collections of actual actions, which would then allow to apply our modularity conditions as they are.

For future work along a different line, our unifying action calculus can be readily used for comparing and assessing action formalisms other than those considered in this paper, and to generalize the specific

translations we have developed to more general classes of domain axiomatizations. Most notably, our approach to abstract from concrete solutions to the Frame Problem should facilitate formal comparisons of the many different existing solutions to the Ramification Problem, thus going beyond comparisons based on specific example scenarios only.

With regard to the result in the second part of the paper, it would be worthwhile for the future to develop a general system for the automatic verification of modularity. By extracting implicit domain constraints, preconditions, and effects from failed attempts to prove the independence conditions, such a system could assist knowledge engineers with the design of "good" axiomatizations. A different line of future work could be the use of the unifying calculus as a method of abstraction for analyzing other problems of general interest across specific calculi.

Acknowledgments.. This paper would have no ground to stand on if it were not for John McCarthy's seminal and ever inspiring work on artificial intelligence, knowledge representation, and commonsense reasoning. I am also deeply grateful for John's continuing interest in my work over the years.

I want to thank the anonymous referees for helpful comments and suggestions on an earlier version of the paper.

References

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

[2] Baker, A. B., 1991. Nonmonotonic reasoning in the framework of situation calculus. Artificial Intelligence 49, 5–23.

[3] Claßen, J., Eyerich, P., Lakemeyer, G., Nebel, B., 2007. Towards an integration of golog and planning. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). AAAI Press, Hyderabad, India.

[4] Doherty, P., Gustafsson, J., Karlsson, L., Kvarnström, J., 1998. Temporal action logics (TAL): Language specification and tutorial. Electronic Transactions on Artificial Intelligence 2 (3–4), 273–306.

[5] Fikes, R. E., Nilsson, N. J., 1971. STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2, 189–208.

[6] Fox, M., Long, D., 2003. PDDL2.1: an extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research 20, 61–124.

[7] Gelfond, M., Lifschitz, V., 1993. Representing action and change by logic programs. Journal of Logic Programming 17, 301–321.

[8] Genesereth, M., Love, N., Pell, B., 2006. General game playing. AI Magazine 26 (2), 73–84.

[9] Ginsberg, M. L., Smith, D. E., 1988. Reasoning about action I: A possible worlds approach. Artificial Intelligence 35, 165–195.

[10] Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., Turner, H., 2004. Nonmonotonic causal theories. Artificial Intelligence 153 (1–2), 49–104.

[11] Gustafsson, J., Doherty, P., 1996. Embracing occlusion in specifying the indirect effects of actions. In: Aiello, L. C., Doyle, J., Shapiro, S. (Eds.), Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR). Morgan Kaufmann, Cambridge, MA, pp. 87–98.

[12] Hanks, S., McDermott, D., 1987. Nonmonotonic logic and temporal projection. Artificial Intelligence 33 (3), 379–412.

[13] Herzig, A., Varzinczak, I., 2004. Domain descriptions should be modular. In: de Mántras, R. L., Saitta, L. (Eds.), Proceedings of the European Conference on Artificial Intelligence (ECAI). IOS Press, pp. 348–352.

[14] Herzig, A., Varzinczak, I., 2005. Cohesion, coupling and the meta-theory of actions. In: Kaelbling, L., Saffiotti, A. (Eds.), Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). Edinburgh, UK, pp. 442–447.

[15] Herzig, A., Varzinczak, I., 2007. Metatheory of actions: beyond consistency. Artificial Intelligence 171 (16–17), 951–984.

[16] Kartha, G. N., 1993. Soundness and completeness theorems for three formalizations of actions. In: Bajcsy, R. (Ed.), Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). Morgan Kaufmann, Chambéry, France, pp. 724–729.

[17] Kowalski, R., Sadri, F., 1994. The situation calculus and event calculus compared. In: Bruynooghe, M. (Ed.), Proceedings of the International Logic Programming Symposium (ILPS). MIT Press, Ithaca, NY, pp. 539–553.

[18] Kowalski, R., Sergot, M., 1986. A logic based calculus of events. New Generation Computing 4, 67–95.

[19] Lin, F., 1995. Embracing causality in specifying the indirect effects of actions. In: Mellish, C. S. (Ed.), Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). Morgan Kaufmann, Montreal, Canada, pp. 1985–1991.

[20] Lin, F., 1996. Embracing causality in specifying the indeterminate effects of actions. In: Clancey, B., Weld, D. (Eds.), Proceedings of the AAAI National Conference on Artificial Intelligence. MIT Press, Portland, OR, pp. 670–676.

[21] McCain, N., Turner, H., 1995. A causal theory of ramifications and qualifications. In: Mellish, C. S. (Ed.), Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). Morgan Kaufmann, Montreal, Canada, pp. 1978–1984.

[22] McCarthy, J., 1963. Situations and Actions and Causal Laws. Stanford Artificial Intelligence Project, Memo 2, Stanford University, CA.
[23] McCarthy, J., 1980. Circumscription—a form of non-monotonic reasoning. Artificial Intelligence 13, 27–39.
[24] McCarthy, J., 1988. Mathematical logic in artificial intelligence. Daedalus 117 (1), 297–311.
[25] McCarthy, J., Hayes, P. J., 1969. Some philosophical problems from the standpoint of artificial intelligence. Machine Intelligence 4, 463–502.
[26] McDermott, D., 2000. The 1998 AI planning systems competition. AI Magazine 21 (2), 35–55.
[27] McIlraith, S., 2000. An axiomatic solution to the ramification problem (sometimes). Artificial Intelligence 116 (1–2), 87–121.
[28] Mueller, E., 2006. Event calculus and temporal action logics compared. Artificial Intelligence 170 (11), 1017–10.
[29] Pednault, E., 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In: Brachman, R., Levesque, H., Reiter, R. (Eds.), Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR). Morgan Kaufmann, Toronto, pp. 324–332.
[30] Pirri, F., Reiter, R., 1999. Some contributions to the metatheory of the situation calculus. Journal of the ACM 46 (3), 261–325.
[31] Reiter, R., 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In: Lifschitz, V. (Ed.), Artificial Intelligence and Mathematical Theory of Computation. Academic Press, pp. 359–380.
[32] Reiter, R., 2001. On knowledge-based programming with sensing in the situation calculus. ACM Transactions on Computational Logic 2 (4), 433–457.
[33] Sandewall, E., 1994. Features and Fluents. The Representation of Knowledge about Dynamical Systems. Oxford University Press.
[34] Sandewall, E., 1994. The range of applicability of some non-monotonic logics for strict inertia. Journal of Logic and Computation 4 (5), 581–615.
[35] Schiffel, S., Thielscher, M., 2006. Reconciling situation calculus and fluent calculus. In: Proceedings of the AAAI National Conference on Artificial Intelligence. AAAI Press, Boston, MA, pp. 287–292.
[36] Shanahan, M., 1995. A circumscriptive calculus of events. Artificial Intelligence 77, 249–284.
[37] Shanahan, M., 1997. Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia. MIT Press.
[38] Shanahan, M., 1999. The ramification problem in the event calculus. In: Dean, T. (Ed.), Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI). Morgan Kaufmann, Stockholm, Sweden, pp. 140–146.
[39] Thielscher, M., 1994. Representing actions in equational logic programming. In: Hentenryck, P. V. (Ed.), Proceedings of the International Conference on Logic Programming (ICLP). MIT Press, Santa Margherita Ligure, Italy, pp. 207–224.
[40] Thielscher, M., 1997. Ramification and causality. Artificial Intelligence 89 (1–2), 317–364.
[41] Thielscher, M., 1999. From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. Artificial Intelligence 111 (1–2), 277–299.
[42] Thielscher, M., 2005. FLUX: A logic programming method for reasoning agents. Theory and Practice of Logic Programming 5 (4–5), 533–565.