1 Motivation

The purpose of the Fluent Calculus is to solve not only the representational but also the inferential Frame Problem. While the former means the problem of specifying all non-effects of actions, the latter concerns the problem of actually inferring these non-effects. When proving a theorem, values of fluents may be needed in situations other than the ones for which they are given or in which they arise as an effect of an action or event. Apparently, one-by-one and using separate instances of the relevant axioms, every such fluent value needs to be carried from the point of its appearance past each intermediate situation to the point of its use. This is done, for instance, in the Situation Calculus if successor state axioms are used, no matter whether reasoning is performed forward in time or via regression [17], and in the Event Calculus where persistence needs to be proven independently for each fluent value [18]. If all fluent values are needed in exactly the situations in which they are given or arise, then the inferential Frame Problem causes no computational burden at all. The more fluents have to be carried unchanged through many intermediate situations or event occurrences, however, the more valuable can a solution to the inferential Frame Problem be.

The Fluent Calculus, which roots in the logic programming formalism of [11], addresses the inferential Frame Problem by specifying the effect of actions in terms of how an action modifies a state. The application of a single **state update axiom** [23] always suffices to infer the entire change caused by the action in question. Central to the axiomatization technique of the Fluent Calculus is a function $\mathit{State}(s)$ which relates a situation s to the state of the world in that situation. In turn, these world states are collections of fluents, which are reified to this end, i.e., treated as terms. That is to say, we use fluent *terms* like $\mathit{On}(A, \mathit{Table})$, where On is a binary function symbol.[1] Fluents that are known to hold in a state are joined together using the binary function symbol "$\circ$". This function is assumed to be both associative and commutative. It is illustratively written in infix notation. Associativity allows us to omit parentheses in nested applications of $\circ$.

As an example, suppose that about the initial situation S_0 in some Blocks World scenario it is known that block A is on some block x, which in turn stands on the table, and that nothing is on top of block A or block B. Using the Fluent Calculus, this incomplete knowledge can be axiomatized by a first-order formula as follows:

$$\begin{array}{l} \exists x, z\, [\, \mathit{State}(S_0) = \mathit{On}(A, x) \circ \mathit{On}(x, \mathit{Table}) \circ z \\ \qquad\qquad \wedge\ \forall y, z'\, [\, z \neq \mathit{On}(y, A) \circ z' \ \wedge\ z \neq \mathit{On}(y, B) \circ z'\,]\,] \end{array} \qquad (1)$$

Put in words, of state $\mathit{State}(S_0)$ it is known that for some x both $\mathit{On}(A, x)$ and $\mathit{On}(x, \mathit{Table})$ are true and possibly some other fluents z hold, too—with the restriction that z does not include a fluent $\mathit{On}(y, A)$ nor a fluent $\mathit{On}(y, B)$, of which we know they are false for any y. This way of axiomatizing negative information relies on a foundational theory of (in-)equality, the *extended unique-name-assumption* (see Section 3).

Following [14], situations are essentially finite sequences of action performances. The function $\mathit{Do}(a, s)$ denotes the situation which results from performing action a in situation s. State update axioms specify how the states at two consecutive situations are related. The universal form of these

[1] A word on the notation: Predicate and function symbols, including constants, start with a capital letter whereas variables are in lower case, sometimes with sub- or superscripts. Free variables in formulas are assumed universally quantified.

axioms is $\Delta(s) \supset \Gamma[State(Do(a,s)), State(s)]$, where $\Delta(s)$ states conditions on s, or rather on the corresponding state, under which Γ defines how the successor state $State(Do(a,s))$ is obtained by modifying the current state $State(s)$.

For example, suppose the effect of an action denoted by $Move(u,v,w)$ is that the block u is moved away from the top of block v onto the top of block w. Let $Poss(a,s)$ represent the property of action a being possible in situation s, then a suitable state update axiom for $Move$ is,

$$\begin{array}{l} Poss(Move(u,v,w),s) \supset \\ \quad State(Do(Move(u,v,w),s)) \circ On(u,v) = State(s) \circ On(u,w) \end{array} \tag{2}$$

That is, if $Move(u,v,w)$ is possible in s, then the state after its performance plus $On(u,v)$ equals the old state plus $On(u,w)$. In other words, the only negative effect of this action is $On(u,v)$ and the only positive effect is $On(u,w)$.

The preconditions of our action $Move(u,v,w)$ are that the block to be relocated, u, is currently on v, that $w \neq u$, and that both u and w are clear, i.e., not obstructed by any other block. Formally,

$$\begin{array}{l} Poss(Move(u,v,w),s) \equiv \\ \quad Holds(On(u,v),s) \wedge w \neq u \wedge \\ \quad \neg\exists x\, [\, Holds(On(x,u),s) \vee Holds(On(x,w),s)\,] \end{array} \tag{3}$$

where $Holds(f,s)$ abbreviates the equational formula $\exists z.\, State(s) = f \circ z$, indicating that f is contained in the state at situation s.

Recall from above the partial initial specification given by formula (1), and suppose block A shall be moved away from its current location onto block B. Then the term $State(S_0)$ in the instance $\{u/A, v/x, w/B, s/S_0\}$ of state update axiom (2) can be replaced by a term which equals $State(S_0)$ according to (1). So doing yields, after evaluating $Poss(Move(A,x,B),S_0)$,

$$\begin{array}{l} \exists x, z\, [\;\; State(Do(Move(A,x,B),S_0)) \circ On(A,x) \;= \\ \qquad\qquad\qquad On(A,x) \circ On(x, Table) \circ z \circ On(A,B) \\ \qquad \wedge\; \forall y, z'.\, z \neq On(y,A)\,] \end{array}$$

The equation can be simplified thus:

$$\begin{array}{l} \exists x, z\, [\;\; State(Do(Move(A,x,B),S_0)) = On(x, Table) \circ z \circ On(A,B) \\ \qquad \wedge\; \forall y, z'.\, z \neq On(y,A)\,] \end{array}$$

We have now obtained from an incomplete initial specification a still partial description of the successor state, which in particular includes both the unaffected fluent $On(x, Table)$ and the information that $On(y,A)$ is false for all y. These two properties of the initial state thus survived the computation of the effect of the action and so need not be carried over by separate applications of one or more axioms.

An example reasoning problem

Providing a solution to the inferential Frame Problem, the merits of state update axioms reveal when longer sequences of actions are considered. The planning problem sketched in Figure 1 is a simple example: Of the starting situation it is known that each block A_i is on top of the corresponding block B_i and that all blocks A_i and C_i are clear. The goal is to reshuffle the configuration so that each block A_i is on the corresponding C_i.

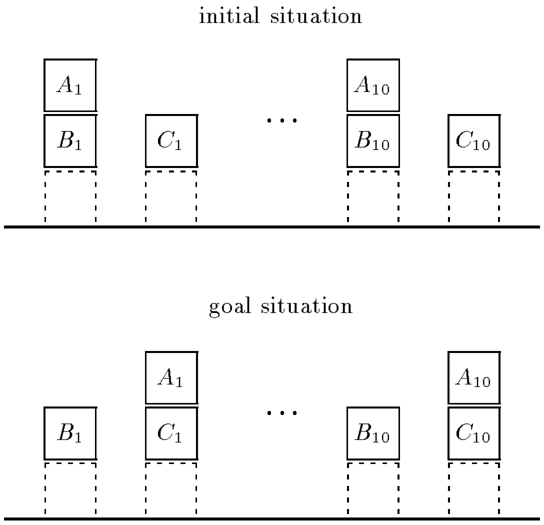

Figure 1: A simple planning problem (with incomplete information).

Let us first encode this planning problem by means of the Situation Calculus formalism of [14] (where fluents are not reified). Let $On(u, v, s)$ denote that block u is on v in situation s, then the partial knowledge of the initial situation is formalized as,

$$\begin{array}{l} On(A_1, B_1, S_0) \wedge \ldots \wedge On(A_{10}, B_{10}, S_0) \wedge \\ \neg\exists x \, [\, On(x, A_1, S_0) \vee \ldots \vee On(x, A_{10}, S_0) \vee \\ \qquad On(x, C_1, S_0) \vee \ldots \vee On(x, C_{10}, S_0) \,] \end{array} \tag{4}$$

The goal is to reach a situation S which satisfies,

$$On(A_1, C_1, S) \wedge \ldots \wedge On(A_{10}, C_{10}, S) \tag{5}$$

Assuming that On is the only relevant fluent and $Move(u, v, w)$ the only relevant action, a suitable effect specification is given by the successor state axiom,

$$\begin{array}{l} Poss(a, s) \supset \\ \quad On(u, w, Do(a, s)) \equiv \exists v.\, a = Move(u, v, w) \\ \qquad\qquad\qquad\qquad\quad \vee\; On(u, w, s) \wedge \forall v.\, a \neq Move(u, w, v) \end{array} \tag{6}$$

along with the precondition axiom,

$$\begin{array}{l} Poss(Move(u, v, w), s) \equiv \\ \quad On(u, v, s) \wedge w \neq u \wedge \neg\exists x \, [\, On(x, u, s) \vee On(x, w, s) \,] \end{array} \tag{7}$$

Now, a straightforward solution to the planning problem is to move in succession the blocks $A_1, \ldots, A_{10}$ away from their initial location onto blocks $C_1, \ldots, C_{10}$, that is,

$$S = Do(Move(A_{10}, B_{10}, C_{10}), \ldots, Do(Move(A_1, B_1, C_1), S_0) \ldots)$$

In order to formally verify that this action sequence is a solution, let UNA be a suitable collection of axioms expressing "uniqueness of names." Then, $\{(6), (7)\} \cup UNA \models (4) \supset (5)$. A proof of this theorem requires at least

190 instances of the successor state axiom, (6).[2] As many as 180 of these instances are used to conclude that some fluent is *not* changed by some action.

The corresponding Fluent Calculus formalization of the planning problem consists of the initial specification,

$$\begin{array}{l} \exists z\, [\, State(S_0) = On(A_1, B_1) \circ \ldots \circ On(A_{10}, B_{10}) \circ z \wedge \\ \quad \forall x, z' \, [\, z \neq On(x, A_1) \circ z' \wedge \ldots \wedge z \neq On(x, A_{10}) \circ z' \wedge \\ \qquad\qquad z \neq On(x, C_1) \circ z' \wedge \ldots \wedge z \neq On(x, C_{10}) \circ z' \,]\,] \end{array} \tag{8}$$

and the goal specification,

$$\exists z.\ State(S) = On(A_1, C_1) \circ \ldots \circ On(A_{10}, C_{10}) \circ z \tag{9}$$

As above, let S be the situation which corresponds to the plan of moving in succession the blocks A_i from B_i onto C_i. Let Ψ be the foundational axioms of the Fluent Calculus (see below), then a proof for the theorem, $\Psi \cup \{(2), (3)\} \models (8) \supset (9)$ requires just 10 instances of the state update axiom, (2), one for each performed action.

The computational value of the Fluent Calculus is crucially dependent on an efficient treatment of equality. While the simple addition of equality axioms may constitute a considerable handicap for theorem proving, a variety of efficient constraint solving algorithms have been developed for the particular equational theory needed for the function $\circ$ (see, e.g., [16] for an overview), in particular for the restricted form of the unification problems which arise in the Fluent Calculus [9].

2 Fluent Calculus Signatures

Fluent Calculus signatures can be considered reified versions of standard Situation Calculus signatures Σ, which are many-sorted first-order languages with equality which include the special sort *sit* for situations [14]. Some predicate symbols in Σ are fluent denotations; these are of arity ≥ 1 with the last argument being of sort *sit*. The corresponding Fluent Calculus signature is then obtained by

1. replacing each $n+1$-place predicate symbol which denotes a fluent and whose argument is of sort $sorts \times sit$ by an n-place function symbol whose argument is of sort *sorts*;
2. adding the binary function symbol "$\circ$" and the constant "$\emptyset$", which serves as a unit element wrt. $\circ$;
3. adding a sort *fluent* to which belong all well-sorted terms with leading function symbol obtained in step 1, and a sort *state* to which belong the constant $\emptyset$, each *fluent*, and each $t_1 \circ t_2$ where t_1, t_2 are of sort *state*;
4. adding the unary function *State*, whose argument is of sort *sit*.

[2] If n is the number of blocks of each kind A, B, and C, then n^2 instances are needed to keep track of the locations of the blocks A_i. Moreover, each action $Move(A_i, B_i, C_i)$ has the preconditions of both A_i and C_i being clear in the situation $Do(Move(A_{i-1}, B_{i-1}, C_{i-1}), \ldots, Do(Move(A_1, B_1, C_1), S_0) \ldots)$, which requires a total of $2 \cdot [(n-1) + (n-2) + \ldots + 1] = n \cdot (n-1)$ additional instances of the successor state axiom.

Fluent Calculus signatures for domains which require a solution to the Ramification Problem include,

5. the unary function " $-$ " whose argument is of sort *fluent*, and the sort *effects* to which belong the constant $\emptyset$, each fluent, each term $-f$ where f is a fluent, and each $t_1 \circ t_2$ where t_1, t_2 are of sort *effects*;

6. the predicate *Causes* of sort $state \times effects \times state \times effects$ and the predicate *Ramify* of sort $state \times effects \times state$.

The meaning and use of this addition will be explained in Section 4.3.

Fluent Calculus signatures for domains which require to reason about the consumption and production of resources include,

7. the sort *resource* such that each *resource* belongs to the sorts *state* and *effects*, which are extended accordingly, and such that the domain of function " $-$ " of item 5 above extends to terms of sort *resource*.

The use of this addition will be explained in Section 5.

In the remainder of this paper, variables of sort *sit* will be denoted by the letter s, variables of sort *fluent* by f, variables of sort *state* by z, variables of sort *effects* by e, and variables of sort *resource* by r, all possibly with sub- or superscripts.

3 Foundational Axioms

Fundamental for any Fluent Calculus axiomatization is the axiom set *EUNA* (the *extended unique-name-assumptions*). Its definition relies on a complete AC1-unification algorithm, i.e., a unification procedure by which are computed complete sets of most general unifiers wrt. the equational theory of associativity, commutativity, and existence of a unit element (see, e.g., [4]). Set *EUNA* comprises the following equational axioms [12].

1. The axioms AC1 for $\circ$ and $\emptyset$,

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

All variables are universally quantified.

2. For any two terms t_1 and t_2 of sort other than *state* or *effects* and with variables $\vec{x}$,

   (a) if t_1 and t_2 are not unifiable, then

   $$\neg \exists \vec{x}.\ t_1 = t_2$$

   (b) if t_1 and t_2 are unifiable with *mgu* θ, then

   $$\forall \vec{x}\ [\ t_1 = t_2 \supset \exists \vec{y}.\ \theta_= \ ]$$

   where $\vec{y}$ denotes the variables which occur in $\theta_=$ but not in $\vec{x}$.[3]

[3] By $\theta_=$ we denote the equational formula $x_1 = r_1 \wedge \ldots \wedge x_n = r_n$ constructed from the substitution $\theta = \{x_1 \mapsto r_1, \ldots, x_n \mapsto r_n\}$.

3. For any two terms t_1 and t_2 of sort *state* or *effects* and with variables $\vec{x}$ and such that function *State* does not occur in t_1 nor in t_2,
   (a) if t_1 and t_2 are not AC1-unifiable, then
   $$\neg \exists \vec{x}.\ t_1 = t_2$$
   (b) if t_1 and t_2 are AC1-unifiable with the complete set of unifiers $cU_{\mathrm{AC1}}(t_1, t_2)$, then
   $$\forall \vec{x} \left[t_1 = t_2 \supset \bigvee_{\theta \in cU_{\mathrm{AC1}}(t_1,t_2)} \exists \vec{y}.\ \theta_= \right]$$
   where $\vec{y}$ denotes the variables which occur in $\theta_=$ but not in $\vec{x}$.

The axioms of item 3, in conjunction with the standard uniqueness of names-assumption in item 2, ensure that *EUNA* is *unification complete* [13, 19] wrt. state terms and the equational theory AC1. These axioms entail inequality of two *state* terms (or *effect* terms, resp.) whenever these are composed of different *fluent* terms.

The assertion that some fluent f holds (resp. does not hold) in some situation s is formalized as $\exists z.\ State(s) = f \circ z$ (resp. $\forall z.\ State(s) \neq f \circ z$). This allows to introduce the common *Holds* predicate, though not as part of the signature but as a mere abbreviation for a certain equality sentence:

$$Holds(f, s) \stackrel{\mathrm{def}}{=} \exists z.\ State(s) = f \circ z \tag{10}$$

Then any Situation Calculus assertion about situations can be easily transferred to the Fluent Calculus; for example, the Situation Calculus formula $On(A, Table, S_0) \vee \forall x. \neg On(x, B, S_0)$ reads $Holds(On(A, Table), S_0) \vee \forall x. \neg Holds(On(x, B), S_0)$ in the Fluent Calculus.

Finally it needs to be guaranteed that state terms do not contain any fluent twice or more, that is,

$$\forall s, f, z.\ State(s) \neq f \circ f \circ z \tag{11}$$

(It will be explained shortly why $\circ$ is not required to be idempotent to this end.)

4 State Update Axioms

The schema $\Delta(s) \supset \Gamma[State(Do(A, s)), State(s)]$ is the universal form of a state update axiom. Typically, condition $\Delta(s)$ combines atom $Poss(A, s)$ with a formula consisting of $Holds(f, s)$ atoms. The form of the update component Γ itself depends on the ontological assumptions that can be made of the action in question. We will discuss three cases in turn.

4.1 The Simple Case

Deterministic actions with only direct and closed effects give rise to the simplest form of state update axioms, where Γ is a mere equation relating $State(Do(A, s))$ to $State(s)$. By closed effects we mean that an action does not have an unbounded number of effects. Suppose action a has a positive effect f, then this fluent simply needs to be coupled onto the old state term via $State(Do(a, s)) = State(s) \circ f$. If action a has a negative effect, then the fluent f which becomes false needs to be withdrawn from the old state. The scheme $State(Do(a,s)) \circ f = State(s)$ serves this purpose.[4] The combination of these two schemes constitutes the general form of state update axioms for deterministic actions with only direct effects:

$$\Delta(s) \supset State(Do(a,s)) \circ \vartheta^{-} = State(s) \circ \vartheta^{+}$$

where ϑ^{-} are the negative effects and ϑ^{+} the positive effects, resp., of action a under condition $\Delta(s)$. The perfect symmetry of the equation in the consequent allows using a state update axiom equally for reasoning forward and backward in time.

State update axiom (2) for the $Move$ action belongs to the simple case; here are two more self-explanatory examples:

$$\begin{array}{l} Poss(Shoot(x,y),s) \wedge Holds(Loaded(x),s) \wedge \neg Holds(Dead(y),s) \supset \\ \quad State(Do(Shoot(x,y),s)) \circ Loaded(x) = State(s) \circ Dead(y) \end{array}$$

$$\begin{array}{l} Poss(Walk(r,x,y),s) \wedge Holds(Time(t),s) \supset \\ \quad State(Do(Walk(r,x,y),s)) \circ At(r,x) \circ Time(t) = \\ \qquad\qquad State(s) \circ At(r,y) \circ Time(t + \frac{Distance(x,y)}{Velocity(r)}) \end{array}$$

Under the provision that actions do have only direct and closed effects, simple state update axioms can be fully mechanically generated from a set of Situation Calculus-style effect axioms if the latter can be assumed to give a complete account of the relevant effects of an action. For example, our state update axiom (2) for the $Move$ action would result from applying this construction to the two effect axioms,

$$\begin{array}{l} Poss(Move(u,v,w),s) \supset Holds(u,w,Do(Move(u,v,w),s)) \\ Poss(Move(u,v,w),s) \supset \neg Holds(u,v,Do(Move(u,v,w),s)) \end{array}$$

It has been proved that a collection of thus generated state update axioms suitably reflects the basic assumption of persistence. This is the **primary theorem** of the Fluent Calculus [23].

4.2 Disjunctive State Update Axioms

Nondeterministic actions are very elegantly specified by means of *disjunctive* state update axioms $\Delta(s) \supset \Gamma[State(Do(a,s)), State(s)]$, where Γ is a disjunction of the possible effects, i.e., state updates, of the respective action. The following, for instance, specifies the alternative outcomes when performing the Russian roulette-like spinning of the chamber of a loaded gun x:

$$\begin{array}{l} Poss(Spin(x),s) \wedge Holds(Loaded(x),s) \supset \\ \qquad State(Do(Spin(x),s)) \circ Loaded(x) = State(s) \\ \qquad \vee \\ \qquad State(Do(Spin(x),s)) = State(s) \end{array}$$

That is, fluent $Loaded(x)$ may or may not become false when performing the action $Spin(x)$.

[4] This scheme is the sole reason for not stipulating that $\circ$ be idempotent, contrary to what one might intuitively expect. For if the function were idempotent, then the equation $State(Do(a,s)) \circ f = State(s)$ would be satisfied if $State(Do(a,s))$ contained f. Hence this equation would not guarantee that f become false. Foundational axiom (11), too, is vital for this scheme since without it the formula $State(Do(a,s)) \circ f = State(s)$ again would not entail $\neg Holds(f, Do(a,s))$ as $State(s)$ could possibly be $f \circ f$.

4.3 State Update Axioms with Ramifications

The Ramification Problem [7] denotes the problem of handling indirect effects of actions. These effects are not explicitly represented in action specifications but follow from general laws, so-called state constraints, describing dependencies among fluents. An example is the extension of the Yale Shooting domain [10] by the state constraint,

$$Holds(Walking(y), s) \supset \neg Holds(Dead(y), s) \tag{12}$$

which is meant to formalize the fact that in all situations all walking things are not dead. As argued in [1], this state constraint gives rise, for instance, to the indirect effect that a turkey stops walking as soon as it is shot. More precisely, if both $Walking(Turkey)$ and $\neg Dead(Turkey)$ happen to be true when an action is performed which causes $Dead(Turkey)$, then this action additionally causes $\neg Walking(Turkey)$.

Such further, indirect effects can be accounted for with the help of *causal relationships* [20, 21]. Each of them defines circumstances under which a single indirect effect is to be expected. A successor state is then the result of applying a chain of causal relationships, after having computed the direct effects of an action.

Let $Causes(z, e, z', e')$ denote that in the current state z the occurred effects e give rise to an additional effect, resulting in the updated state z' and the updated current effects e'. For instance, the (only) kind of indirect effects in our example is accommodated via the following definition:

$$\begin{aligned} Causes(z, e, z', e') \equiv\; & \exists z.\ e = Dead(y) \circ z \wedge \\ & z' \circ Walking(y) = z \wedge \\ & e' = e \circ -Walking(y) \end{aligned} \tag{13}$$

where a sub-term $-F$ represents the occurrence of a negative effect. Put in words, if $Dead(y)$ occurs as (direct or indirect) effect, then this causes $Walking(y)$ to become false, in z', as indirect effect, which also yields an extended collection of current effects, e'. From (13) we can derive, for instance, that whenever the turkey is dead but still walking after an action has occurred with the effects $-Loaded(Gun)$ and $Dead(Turkey)$, then $-Walking(Turkey)$ is additionally caused; that is, formally,

$$\begin{aligned} Causes(& Dead(Turkey) \circ Walking(Turkey) \circ z, \\ & -Loaded(Gun) \circ Dead(Turkey), \\ & Dead(Turkey) \circ z, \\ & -Loaded(Gun) \circ Dead(Turkey) \circ -Walking(Turkey)) \end{aligned}$$

State update axioms which account for indirect effects are of the form,

$$\begin{aligned} & \Delta(s) \supset \\ & \quad z \circ \vartheta^- = State(s) \circ \vartheta^+ \supset \\ & \qquad Ramify(z, -\vartheta^- \circ \vartheta^+, State(Do(a, s))) \end{aligned} \tag{14}$$

where

- ϑ^- are the direct negative effects;
- ϑ^+ are the direct positive effects;
- $Ramify(z, e, z_{new})$ means that state z_{new} is reachable by successively applying (zero or more) causal relationships to state z and effects e.

Usually, most states z_{new} which are reachable from z, e violate the underlying state constraints and, hence, can only be intermediate states on the way to a possible successor state: Schema (14) says that we are interested in reaching a state which can be assigned to the expression $State(Do(a, s))$, hence which satisfies all constraints (c.f. (12), e.g., which quantifies over all situations and therefore applies to all states assigned to a situation).

The definition of predicate *Ramify* requires a standard second-order axiom to characterize the reflexive and transitive closure of *Causes*:

$$Ramify(z, e, z_{new}) \equiv$$
$$\forall \Pi \left\{ \begin{array}{c} \forall z_1, e_1 .\ \Pi(z_1, e_1, z_1, e_1) \\ \wedge \\ \left[\begin{array}{l} \forall z_1, e_1, z_2, e_2, z_3, e_3 . \\ \quad \Pi(z_1, e_1, z_2, e_2) \wedge Causes(z_2, e_2, z_3, e_3) \supset \Pi(z_1, e_1, z_3, e_3) \end{array} \right] \\ \supset \\ \exists e_{new} .\ \Pi(z, e, z_{new}, e_{new}) \end{array} \right\}$$

Along with the axioms above and the foundational axioms of the basic Fluent Calculus, this state update axiom,

$$\begin{array}{l} Poss(Shoot(x, y), s) \wedge Holds(Loaded(x), s) \wedge \neg Holds(Dead(y), s) \supset \\ \quad z \circ Loaded(x) = State(s) \circ Dead(y) \supset \\ \quad\quad Ramify(\, z, -Loaded(x) \circ Dead(y), State(Do(Shoot(x, y), s))) \end{array}$$

entails that $Holds(Loaded(Gun), S_0) \supset \neg Holds(Walking(Turkey), S_1)$, with $S_1 = Do(Shoot(Gun, Turkey), S_0))$.

5 Reasoning about Resources

The Fluent Calculus offers a most natural way of reasoning about the dynamic production and consumption of resources. This feature is rooted in the non-idempotency of the connection function $\circ$, which implies that *EUNA* entails $t_1 \neq t_2$ whenever any sub-term which is not the unit element $\emptyset$ occurs a different number of times in t_1 than in t_2. State constituents may thus not only represent fluents but also resources, for which the number of occurrences matters. The following, for example, holds true under *EUNA*:

$$Wheel(d, x) \circ Wheel(d, x) \circ Axle(l, y) \neq Wheel(d, x) \circ Axle(l, y) \circ Axle(l, y)$$

This inequation can be read as, having available two wheels (with identical diameter d and a hole of diameter x in the center) and one axle (of length l and diameter y) is different from having available just one such wheel but two axles.

The reader may recall foundational axiom (11), by which state terms associated to situations are required to contain at most one occurrence of a fluent. Neither this nor a similar restriction applies to resources.

As an example, let $Wheel(d, x)$, $Axle(l, y)$, and $Chassis(l, d)$ be of sort *resource*. Then the following formula states the initial availability of at least two axles of length $3.5'$ and the availability of exactly three wheels of identical diameter such that the two axles fit into the center holes of the wheels:

$$\begin{array}{l} \exists d, x, z\, [\; State(S_0) = Axle(3.5', x) \circ Axle(3.5', x) \circ \\ \qquad\qquad\qquad\qquad Wheel(d, x) \circ Wheel(d, x) \circ Wheel(d, x) \circ z \\ \qquad\quad \wedge\ \forall d', x', z' .\ z \neq Wheel(d', x') \circ z' \,] \end{array}$$

Consider the action of assembling a two-wheel chassis of width l and diameter d, denoted by $Assemble(l, d)$. This action is possible iff two wheels of diameter d and a fitting axle of length l are available:

$$\begin{array}{l} Poss(Assemble(l,d),s) \equiv \\ \quad \exists x.\, Holds(Wheel(d,x) \circ Wheel(d,x) \circ Axle(l,x), s) \end{array}$$

In this formula we have employed a generalization of the *Holds* macro (c.f. formula (10)) which allows the first argument to be of sort *state*:

$$Holds(z_1, s) \stackrel{\text{def}}{=} \exists z.\, State(s) = z_1 \circ z$$

The effect of assembling a chassis is that two wheels and one axle are 'consumed' and a corresponding chassis is produced. Formally,

$$\begin{array}{l} Poss(Assemble(l,d),s) \supset \\ \quad \exists x.\, State(Do(Assemble(l,d),s)) \circ \\ \qquad\qquad Wheel(d,x) \circ Wheel(d,x) \circ Axle(l,x) = State(s) \circ Chassis(l,d) \end{array}$$

The axioms of this section along with the foundational axioms of the Fluent Calculus entail, for instance, that in the initial situation it is possible to assemble one chassis but—due to the limited supply of wheels—not more:

$$\begin{array}{l} \exists d.\, Poss(Assemble(\mathtt{3.5'}, d), S_0) \land \\ \neg\exists l, d.\, Poss(Assemble(l,d), Do(Assemble(l,d), S_0)) \end{array}$$

6 Historical and Bibliographical Remarks

The Fluent Calculus roots in the equational logic programming formalism of [11], which introduced the binary connection function "$\circ$" along with the principle of modeling change as rewriting collections of fluent terms. The monograph [22] contains a detailed biographical account of the research based on the principles which underlie the Fluent Calculus. Most notably, in [8] the original formalism was proved to be closely related to earlier approaches to the Frame Problem which appeal to non-classical logics, namely, linearized versions of the connection method [2] (now also known as Transition Logic [3]) and of Gentzen's sequent calculus [15], both of which embed into a logical framework the operational STRIPS approach [5]. A detailed account of these two formalisms, including a number of further comparison results, is given in the monograph [6].

References

[1] Andrew B. Baker. Nonmonotonic reasoning in the framework of situation calculus. *Artificial Intelligence*, 49:5–23, 1991.

[2] Wolfgang Bibel. A deductive solution for plan generation. *New Generation Computing*, 4:115–132, 1986.

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

[4] Hans-Jürgen Bürckert, Alexander Herold, Deepak Kapur, Jörg H. Siekmann, Mark E. Stickel, M. Tepp, and H. Zhang. Opening the AC-unification race. *Journal of Automated Reasoning*, 4:465–474, 1988.

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

[6] Bertram Fronhöfer. *The Action-as-Implication Paradigm*. CS Press München, 1996.

[7] Matthew L. Ginsberg and David E. Smith. Reasoning about action II: The qualification problem. *Artificial Intelligence*, 35:311–342, 1988.

[8] Gerd Große, Steffen Hölldobler, and Josef Schneeberger. Linear Deductive Planning. *Journal of Logic and Computation*, 6(2):233–262, 1996.

[9] Gerd Große, Steffen Hölldobler, Josef Schneeberger, Ute Sigmund, and Michael Thielscher. Equational logic programming, actions, and change. In K. Apt, editor, *Proceedings of the International Joint Conference and Symposium on Logic Programming (IJCSLP)*, pages 177–191, Washington, 1992. MIT Press.

[10] Steve Hanks and Drew McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3):379–412, 1987.

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

[12] Steffen Hölldobler and Michael Thielscher. Computing change and specificity with equational logic programs. *Annals of Mathematics and Artificial Intelligence*, 14(1):99–133, 1995.

[13] Joxan Jaffar, Jean-Louis Lassez, and Michael J. Maher. A theory of complete logic programs with equality. *Journal of Logic Programming*, 1(3):211–223, 1984.

[14] Hector Levesque, Fiora Pirri, and Ray Reiter. Foundations for the situation calculus. *Linköping Electronic Articles in Computer and Information Science*, 3(18), 1998. URL: `http://www.ep.liu.se/ea/cis/1998/018/`.

[15] M. Masseron, Christophe Tollu, and Jacqueline Vauzielles. Generating plans in linear logic I. Actions as proofs. *Journal of Theoretical Computer Science*, 113:349–370, 1993.

[16] Leszek Pacholski and Andreas Podelski. Set constraints: a pearl in research on constraints. In G. Smolka, editor, *Proceedings of the International Conference on Constraint Programming (CP)*, volume 1330 of *LNCS*, pages 549–561. Springer, 1997.

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

[18] Murray Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press, 1997.

[19] John C. Shepherdson. SLDNF-resolution with equality. *Journal of Automated Reasoning*, 8:297–306, 1992.

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

[21] Michael Thielscher. Reasoning about actions: Steady versus stabilizing state constraints. *Artificial Intelligence*, 104:339–355, 1998.

[22] Michael Thielscher. *Challenges for Action Theories: Solving the Ramification and Qualification Problem.* Springer LNAI, 1999.

[23] Michael Thielscher. From Situation Calculus to Fluent Calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence*, 1999.