Linköping Electronic Articles in
Computer and Information Science
Vol. 3(1998): nr 14

Introduction To The Fluent Calculus

Michael Thielscher

Department of Computer Science
Dresden University of Technology
Dresden, Germany

Linköping University Electronic Press
Linköping, Sweden

http://www.ep.liu.se/ea/cis/1998/14/

Published on October 4, 1998 by
Linköping University Electronic Press
581 83 Linköping, Sweden

Linköping Electronic Articles in
Computer and Information Science
ISSN 1401-9841
Series editor: Erik Sandewall

Typeset by the author using LaTeX
Formatted using étendu *style*

Recommended citation:
< Author >. < Title >. Linköping Electronic Articles in Computer and Information Science, Vol. 3(1998): nr 14. http://www.ep.liu.se/ea/cis/1998/14/. *October 4, 1998.*

This URL will also contain a link to the author's home page.

1 Motivation

The purpose of the Fluent Calculus is to solve the inferential Frame Problem. While the representational aspect of the Frame Problem means the problem of specifying all non-effects of actions, the inferential aspect means the problem of actually computing these non-effects. The latter concerns each fluent value which, when proving a theorem, is needed in a situation other than the one for which it is given or in which it arises 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 [10], and in the Event Calculus where survival needs to be proven independently for each fluent value [11]. 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 approach of [6] and was christened in [2], addresses the inferential Frame Problem by specifying the effect of actions in terms of how an action modifies a state. A single so-called state update axiom always suffices to derive the entire change caused by the action in question. Central to this axiomatization technique is a function $State(s)$ 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. 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. As an example, suppose that about the initial state 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 B. In the Fluent Calculus, this incomplete knowledge can be axiomatized as follows:

$$\begin{aligned} \exists x, z.\ & State(S_0) = On(A,x) \circ On(x, Table) \circ z \\ & \wedge \forall y, z'.\ z \neq On(y,B) \circ z' \end{aligned} \tag{1}$$

Put in words, of state $State(S_0)$ it is known that for some x both $On(A,x)$ and $On(x, Table)$ are true and possibly some other facts z hold, too—with the restriction that z does not include a fluent of the form $On(y,B)$.

State update axioms specify the entire relation between the states at two consecutive situations. The universal form of these 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, let the effect of an action called $Move(u,v,w)$ be that block u is moved from the top of block v on top of block w. A suitable state update axiom is,

$$\begin{aligned} & Poss(Move(u,v,w),s) \supset \\ & \quad State(Do(Move(u,v,w),s)) \circ On(u,v) = State(s) \circ On(u,w) \end{aligned}$$

That is, if $Move(u,v,w)$ is possible and performed in s, then the new state 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 $On(u,w)$.

Suppose for the sake of argument that $Poss(Move(A,x,B),S_0)$ be given. Then the expression $State(S_0)$ in the appropriately instantiated state update axiom can be replaced by the term which equals $State(S_0)$ according to formula (1). So doing yields,

$$\begin{aligned}\exists x,z.\ State(Do(Move(A,x,B),S_0)) \circ On(A,x) = \\ On(A,x) \circ On(x,Table) \circ z \circ On(A,B)\end{aligned}$$

This formula can be simplified to

$$\exists x,z.\ State(Do(Move(A,x,B),S_0)) = On(x,Table) \circ z \circ On(A,B)$$

In this way one obtains from an incomplete initial specification a still partial description of the successor state, which in particular includes the unaffected fluent $On(x,Table)$. This property thus survived the computation of the effect of the action and so needs not be carried over by separate application of an axiom.

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., [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 including the sort sit for situations. 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 $state$, which is the least set 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 binary predicate $Holds$, whose argument is of sort $fluent \times sit$, and the unary function $State$, whose argument is of sort sit.

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 (see, e.g., [3]). Set $EUNA$ comprises the following equational axioms [7].

1. The axioms AC1 (i.e., associativity, commutativity, unit element) and the standard equality axioms, that is,

$$
\begin{array}{l}
(x \circ y) \circ z = x \circ (y \circ z) \\
x \circ y = y \circ x \\
x \circ \emptyset = x \\
x = x \\
x = y \;\supset\; y = x \\
x = y \wedge y = z \;\supset\; x = z \\
x_i = y \;\supset\; f(x_1, \ldots, x_i, \ldots, x_n) = f(x_1, \ldots, y, \ldots, x_n) \\
x_i = y \;\supset\; [P(x_1, \ldots, x_i, \ldots, x_n) \equiv P(x_1, \ldots, y, \ldots, x_n)
\end{array}
$$

   for each n-place function symbol f and predicate P, and for each $1 \leq i \leq n$. All variables are universally quantified.

2. For any two terms t_1 and t_2 of sort other than *state* 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 $\vec{x}$.[1]

3. For any two terms t_1 and t_2 of sort *state* and with variables $\vec{x}$,

   (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* [8, 12] wrt. state terms and the equational theory AC1. These axioms entail inequality of two state terms 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 $\exists z.\ \mathit{State}(s) = f \circ z$ (resp. $\forall z.\ \mathit{State}(s) \neq f \circ z$). This allows to introduce the common *Holds* predicate, though not as a primitive notion but as a derived concept:

$$\mathit{Holds}(f, s) \;\equiv\; \exists z.\ \mathit{State}(s) = f \circ z \tag{2}$$

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

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

$$\forall s, x, z.\ \mathit{State}(s) = x \circ x \circ z \supset x = \emptyset \tag{3}$$

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

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 potentially infinitely many 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.[2] 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, respectively, 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.

Here are two self-explanatory examples of simple state update axioms:

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

$$\begin{array}{l} Poss(Walk(r,x,y),s) \wedge Holds(Time(t),s) \wedge t' = t + \frac{Distance(x,y)}{Velocity(r)} \supset \\ \quad State(Do(Walk(r,x,y),s)) \circ At(r,x) \circ Time(t) = \\ \qquad State(s) \circ At(r,y) \circ Time(t') \end{array}$$

4.2 Disjunctive State Update Axioms

Nondeterministic actions can be 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:

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

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

[2] This scheme is the sole reason for not stipulating that "$\circ$" be idempotent, contrary to what the reader might have expected. 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 (3), too, is vital for this scheme in case of incomplete knowledge of word states.

4.3 State Update Axioms with Ramifications

The Ramification Problem [4] 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. As an example, consider the extension of the Yale Shooting domain [5] by the state constraint $Walking(y) \supset \neg Dead(y)$. The constraint itself is straightforwardly axiomatized as

$$Holds(Walking(y), s) \supset \neg Holds(Dead(y), s) \tag{4}$$

As argued in [1], this state constraint gives rise to the indirect effect that the 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 by successive application of so-called causal relationships [13]. Their axiomatization is based on a predicate $Causes(state, effects, new\_state, new\_effects)$, which means that in the current state $state$ the occurred effects $effects$ give rise to an additional effect resulting in the updated state new_state and the updated current effects $new\_effects$. In this way, the indirect effect in the example is accommodated via the following definition:

$$\begin{aligned} Causes(state, effects, new\_state, new\_effects) \equiv & \\ & \exists x.\ effects = Dead(y) \circ x \wedge \\ & new\_state \circ Walking(y) = state \wedge \\ & new\_effects = effects \circ -Walking(y) \end{aligned}$$

where a sub-term $-F$ represents the occurrence of a negative effect. From this definition 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

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

where

- ϑ^- are the negative direct effects;
- ϑ^+ are the positive direct effects;
- $Ramify(state, effects, new\_state)$ means that the successive application of (zero or more) causal relationships to state $state$ and effects $effects$ results in state new_state.

The axiomatizations of the underlying state constraints, such as (4), guarantee that the overall resulting state, $State(Do(A, s))$, satisfies all constraints.

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

$$
\begin{array}{l}
Ramify(state, effects, new\_state) \equiv \\
\forall \Pi \left\{ \begin{array}{c}
\forall s_1, e_1.\ \Pi(s_1, e_1, s_1, e_1) \\
\wedge \\
\left[\begin{array}{l}
\forall s_1, e_1, s_2, e_2, s_3, e_3. \\
\quad \Pi(s_1, e_1, s_2, e_2) \wedge Causes(s_2, e_2, s_3, e_3) \\
\qquad \supset \Pi(s_1, e_1, s_3, e_3)
\end{array} \right] \\
\supset \\
\exists new\_effects.\ \Pi(state, effects, new\_state, new\_effects)
\end{array} \right\}
\end{array}
$$

Along with the axioms above, this state update axiom,

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

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

References

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

[2] Sven-Erik Bornscheuer and Michael Thielscher. Explicit and implicit indeterminism: Reasoning about uncertain and contradictory specifications of dynamic systems. *Journal of Logic Programming*, 31(1–3):119–155, 1997.

[3] 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.

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

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

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

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

[8] 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.

[9] 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.

[10] 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.

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

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

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