

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/1998/014/>

Revised version

*Revised version, published on February 10 by
Linköping University Electronic Press
581 83 Linköping, Sweden
Original version was published on October 4, 1998*

**Linköping Electronic Articles in
Computer and Information Science**

ISSN 1401-9841

Series editor: Erik Sandewall

*©1998 Michael Thielscher
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/1998/014/> . October 4, 1998.*

*The URL will also contain links to both the original version and
the present revised version, as well as to the author's home page.*

*The publishers will keep this article on-line on the Internet
(or its possible replacement network in the future)
for a period of 25 years from the date of publication,
barring exceptional circumstances as described separately.*

*The on-line availability of the article implies
a permanent permission for anyone to read the article on-line,
to print out single copies of it, and to use it unchanged
for any non-commercial research and educational purpose,
including making copies for classroom use.*

*This permission can not be revoked by subsequent
transfers of copyright. All other uses of the article are
conditional on the consent of the copyright owner.*

*The publication of the article on the date stated above
included also the production of a limited number of copies
on paper, which were archived in Swedish university libraries
like all other written works published in Sweden.
The publisher has taken technical and administrative measures
to assure that the on-line version of the article will be
permanently accessible using the URL stated above,
unchanged, and permanently equal to the archived printed copies
at least until the expiration of the publication period.*

*For additional information about the Linköping University
Electronic Press and its procedures for publication and for
assurance of document integrity, please refer to
its WWW home page: <http://www.ep.liu.se/>
or by conventional mail to the address stated above.*

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 logic programming formalism of [5], 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** [15] always suffices to derive the entire change caused by the action in question. Central to the axiomatization technique of the Fluent Calculus 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 A or block B . In the Fluent Calculus, this incomplete knowledge can be axiomatized as follows:

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

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 $On(y, A)$ nor $On(y, B)$, of which we know they are false.

State update axioms specify how the states at two consecutive situations are related to each other. 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 denoted by $Move(u, v, w)$ be that the block u is moved away from the top of block v onto the top of block w . A suitable state update axiom is,

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

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

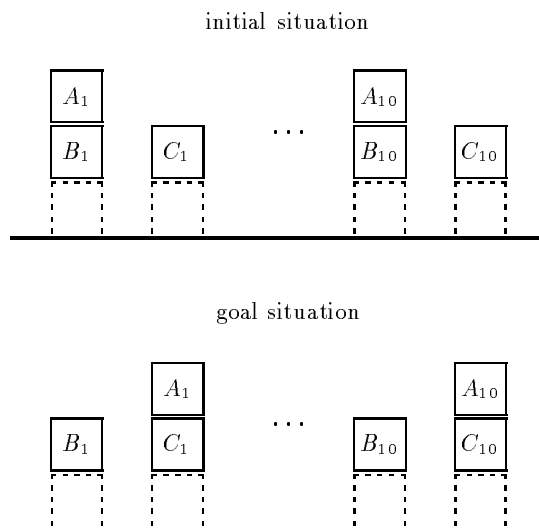


Figure 1: A simple planning problem (with incomplete information).

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 and that both u and w are clear, i.e., not obstructed by any other block. Formally,

$$Poss(Move(u, v, w), s) \equiv Holds(On(u, v), s) \wedge \neg \exists x [Holds(On(x, u), s) \vee Holds(On(x, w), s)] \quad (3)$$

where $Holds(f, s)$ is a macro which abbreviates $\exists z. State(s) = f \circ z$.

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 $Poss(Move(A, x, B), S_0)$,

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

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.

Providing a solution to the inferential Frame Problem, the merits of state update axioms become apparent as soon as larger reasoning problems are considered. Take, for example, the planning problem sketched in Figure 1: 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 .

We first encode this planning problem by means of the Situation Calculus formalism as described by [8]. Let $On(u, v, s)$ denote that block u is on v

in situation s , then the partial knowledge of the initial situation can be formalized as,

$$\begin{aligned} & On(A_1, B_1, S_0) \wedge \dots \wedge On(A_{10}, B_{10}, S_0) \wedge \\ & \neg \exists x [On(x, A_1, S_0) \vee \dots \vee On(x, A_{10}, S_0) \vee \\ & \quad On(x, C_1, S_0) \vee \dots \vee On(x, C_{10}, S_0)] \end{aligned} \quad (4)$$

The goal is to reach a situation S which satisfies,

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

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,

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

along with the precondition axiom,

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

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

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

In order to formally verify this action sequence 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 145 instances of the successor state axiom, (6).¹ As many as 135 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,

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

and the goal specification,

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

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., [9].

¹ 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 , and $(n-1) + (n-2) + \dots + 1 = \frac{n}{2} \cdot (n-1)$ instances for the relevant information about the blocks C_i not (yet) being occupied.

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 [8]. 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 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, 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 unit element (see, e.g., [2]). Set *EUNA* comprises the following equational axioms [6].

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

$$\begin{aligned} (x \circ y) \circ z &= x \circ (y \circ z) \\ x \circ y &= y \circ x \\ x \circ \emptyset &= x \end{aligned}$$

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} .²

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

² By $\theta_{=}$ we denote the equational formula $x_1 = r_1 \wedge \dots \wedge x_n = r_n$ constructed from the substitution $\theta = \{x_1 \mapsto r_1, \dots, x_n \mapsto r_n\}$.

(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 $cU_{AC1}(t_1, t_2)$, then

$$\forall \vec{x} \left[t_1 = t_2 \supset \bigvee_{\theta \in cU_{AC1}(t_1, t_2)} \exists \vec{y}. \theta = \right]$$

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* [7, 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 $\exists z. State(s) = f \circ z$ (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:

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

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,

$$\forall s, x, z. State(s) = x \circ x \circ z \supset x = \emptyset \quad (10)$$

(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 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.³ 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.

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

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

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

Under the provision that actions do not have open effects,⁴ simple state update axioms can be fully mechanically generated from a set of Situation Calculus-style effect axioms if the latter can be assumed a complete account of the relevant effects of an action [15]. For example, our state update axiom (2) for the *Move* action would result from applying this construction to the two effect axioms,

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

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 [15].

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 :

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

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

³ 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 (10), too, is vital for this scheme in case of incomplete knowledge of world states.

⁴ By open effects we mean an unbounded number of direct effects.

4.3 State Update Axioms with Ramifications

The Ramification Problem [3] 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 [4] by the state constraint $Walking(y) \supset \neg Dead(y)$. The constraint itself is straightforwardly axiomatized as,

$$Holds(Walking(y), s) \supset \neg Holds(Dead(y), s) \quad (11)$$

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 the successive application of so-called causal relationships [13, 14]. These are used to define 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:

$$\begin{aligned} Causes(state, effects, new_state, new_effects) \subset \\ \exists z. effects = Dead(y) \circ z \wedge \\ new_state \circ Walking(y) = state \wedge \\ new_effects = effects \circ -Walking(y) \end{aligned}$$

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,

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

where

- ϑ^- are the direct negative effects;
- ϑ^+ are the direct positive 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 (11), guarantee that the overall resulting state, $State(Do(A, s))$, satisfies all con-

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

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

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

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

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] Hans-Jürgen Bürkert, 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.
- [3] Matthew L. Ginsberg and David E. Smith. Reasoning about action II: The qualification problem. *Artificial Intelligence*, 35:311–342, 1988.
- [4] Steve Hanks and Drew McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3):379–412, 1987.
- [5] Steffen Hölldobler and Josef Schneeberger. A new deductive approach to planning. *New Generation Computing*, 8:225–244, 1990.
- [6] Steffen Hölldobler and Michael Thielscher. Computing change and specificity with equational logic programs. *Annals of Mathematics and Artificial Intelligence*, 14(1):99–133, 1995.
- [7] 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.
- [8] 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/>.
- [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.
- [14] Michael Thielscher. Reasoning about actions: Steady versus stabilizing state constraints. *Artificial Intelligence*, 104:339–355, 1998.
- [15] Michael Thielscher. Towards state update axioms: Reifying successor state axioms. In L. F. del Cerro, J. Dix, and U. Furbach, editors, *Proceedings of the European Workshop on Logics in AI (JELIA)*, volume 1489 of *LNAI*, pages 248–263, Dagstuhl, Germany, October 1998. Springer. (Also conditionally accepted for *Artificial Intelligence*).