The Common Core of Action Languages $\mathcal{B}$ and $\mathcal{C}$

Michael Gelfond
Texas Tech University

Vladimir Lifschitz
University of Texas at Austin

Abstract

Action languages $\mathcal{B}$ and $\mathcal{C}$ are similar to each other in the sense that each of them is capable of describing indirect effects of actions. On the other hand, these languages are not equally expressive: each of them has its own distinctive features that cannot be easily translated into the other language. We clarify the relationship between the expressive capabilities of these languages by describing their common core—a subset of $\mathcal{B}$ that can be translated into $\mathcal{C}$ by a simple syntactic transformation.

Introduction

Action languages $\mathcal{B}$ (Gelfond & Lifschitz 1998, Section 5) and $\mathcal{C}$ (Giunchiglia & Lifschitz 1998), (Gelfond & Lifschitz 1998, Section 6) differ from the previous generation of action description languages[1] in that they allow us to describe ramifications, or indirect effects, of executing an action. This is achieved by introducing constructs that represent relationships between fluents. For instance, walking to a different place affects not only the person's location but also, indirectly, the location of the cell phone in his pocket, because of a relationship between his own location and the location of the phone. Such relationships can be called static, because they operate within a single time instant. Action descriptions in older languages were limited to dynamic relationships between an action and the values of fluents after its execution.

The languages $\mathcal{B}$ and $\mathcal{C}$ differ from each other in the sense that they are based on different intuitions about causality, and also in the sense that they are not equally expressive: each of them has its own distinctive features that cannot be easily translated into the other language. In this note, we clarify the relationship between the expressive capabilities of these languages by describing their common core—a subset of $\mathcal{B}$ that can be translated into $\mathcal{C}$ by a simple syntactic transformation. This transformation turns a $\mathcal{B}$-description into its "$\mathcal{C}$-image" by appending a standard set of postulates formalizing the commonsense law of inertia (Shanahan 1997). It is needed because the commonsense law of intertia is built into the semantics of $\mathcal{B}$; in $\mathcal{C}$, it is easily expressible, but not automatically included.

To guarantee the equivalence of a $\mathcal{B}$-description to its $\mathcal{C}$-image we need to assume that its static laws are free of cycles. Conditions of this type have been used in the study of equivalent transformations in the logic of universal causation (Turner 1998, Theorem 5.15) and for the purpose of simplifying logic programming representations of action domains described in $\mathcal{C}$ (Lifschitz & Turner 1999, Proposition 2).

[1] STRIPS (Fikes & Nilsson 1971); ADL (Pednault 1989); $\mathcal{A}$ (Gelfond & Lifschitz 1993).

Review: Languages $\mathcal{B}$ and $\mathcal{C}$

We begin with a finite set of propositional atoms divided into two groups, *fluents* and *elementary actions*. An *action* is a function from elementary actions to truth values. A *transition system* T is determined by a set of functions from fluents to truth values, called the *states* of T, and a set of triples $\langle s_0, a, s_1\rangle$, where s_0 and s_1 are states of T, and a is an action. These triples are called the *transitions* of T. A transition system can be visualized as a directed graph that has states as its vertices, with an edge from s_0 to s_1 labeled a for every transition $\langle s_0, a, s_1\rangle$.

Language $\mathcal{B}$

Syntax A *fluent literal* is a literal containing a fluent. A *condition* is a set of fluent literals.

An action description in the language $\mathcal{B}$, or $\mathcal{B}$-*description*, is a set of expressions of the following two forms:

- *static laws*

$$l \textbf{ if } c, \tag{1}$$

  where l is a fluent literal, and c is a condition;

- *dynamic laws*

$$e \textbf{ causes } l \textbf{ if } c, \tag{2}$$

  where e is an elementary action, l is a fluent literal, and c is a condition.

Semantics We will identify a function assigning truth values to atoms with the set of literals that get the value *true*. In particular, any action can be thought of as a set of elementary actions and their negations, and any

state of a transition system can be thought of as a set of fluent literals.

About a set X of literals we say that it is *closed* under a $\mathcal{B}$-description D if, for every static law (1) from D, $l \in X$ whenever $c \subseteq X$. By $Cn_D(X)$ ("consequences of X under D") we denote the smallest set of literals that contains X and is closed under D.

For any $\mathcal{B}$-description D, the transition system $T(D)$ *represented by* D is defined as follows:

- the states of $T(D)$ are the functions from fluents to truth values that are closed under D;
- $\langle s_0, a, s_1\rangle$ is a transition of $T(D)$ iff

$$s_1 = Cn_D(X \cup (s_0 \cap s_1)), \tag{3}$$

where X is the set of all literals l such that, for some dynamic law (2) from D, $e \in a$ and $c \subseteq s_0$.

Discussion We understand a static law (1) as the inference rule allowing us to derive the new fact l from the facts c established earlier. On the right-hand side of the McCain-Turner equation (3), X is the set of explicit effects of a, $s_0 \cap s_1$ is the set of facts justified by inertia, and the application of Cn_D generates the indirect effects of a by applying the inference rules expressed by the static laws.

Example 1. D consists of the dynamic law

$$e \textbf{ causes } q \textbf{ if } p$$

and the static law

$$\neg r \textbf{ if } q.$$

In this case the states of $T(D)$ are

$$\begin{array}{lll} \{p, q, \neg r\}, & \{p, \neg q, r\}, & \{p, \neg q, \neg r\}, \\ \{\neg p, q, \neg r\}, & \{\neg p, \neg q, r\}, & \{\neg p, \neg q, \neg r\}. \end{array}$$

The transitions of $T(D)$ can be divided into three groups. In some, e is executed and the precondition p of its effect on q is satisfied:

$$\begin{array}{c} \langle\{p, q, \neg r\}, \{e\}, \{p, q, \neg r\}\rangle, \\ \langle\{p, \neg q, r\}, \{e\}, \{p, q, \neg r\}\rangle, \\ \langle\{p, \neg q, \neg r\}, \{e\}, \{p, q, \neg r\}\rangle. \end{array} \tag{4}$$

(In the second of transitions (4), r is indirectly affected.) In others, e is executed but the precondition p is not satisfied:

$$\begin{array}{c} \langle\{\neg p, q, \neg r\}, \{e\}, \{\neg p, q, \neg r\}\rangle, \\ \langle\{\neg p, \neg q, r\}, \{e\}, \{\neg p, \neg q, r\}\rangle, \\ \langle\{\neg p, \neg q, \neg r\}, \{e\}, \{\neg p, \neg q, \neg r\}\rangle. \end{array}$$

Finally, there are 6 transitions in which e is not executed: $\langle s, \{\neg e\}, s\rangle$ for all states s of $T(D)$.

To check, for instance, that the second of transitions (4) satisfies equation (3), we calculate:

$$\begin{array}{l} Cn_D(X \cup (s_0 \cap s_1)) \\ \quad = Cn_D(\{q\} \cup (\{p, \neg q, r\} \cap \{p, q, \neg r\})) \\ \quad = Cn_D(\{p, q\}) = \{p, q, \neg r\} = s_1. \end{array}$$

Language $\mathcal{C}$

Syntax A *formula* is a propositional combination of atoms. *An action description in the language* $\mathcal{C}$, or $\mathcal{C}$*-description*, is a set of expressions of the following two forms:

- *static laws*

$$\textbf{caused } f \textbf{ if } g, \tag{5}$$

where f and g are formulas that do not contain elementary actions;

- *dynamic laws*

$$\textbf{caused } f \textbf{ if } g \textbf{ after } h, \tag{6}$$

where f and g are as above, and h is a formula.

Semantics By $\models$ we denote the satisfaction relation of classical propositional logic.

For any $\mathcal{C}$-description D and any transition $\langle s_0, a, s_1\rangle$, $D^{\langle s_0, a, s_1\rangle}$ ("the reduct of D relative to $\langle s_0, a, s_1\rangle$") stands for the set consisting of

- the formulas f for all static laws (5) from D such that $s_1 \models g$, and
- the formulas f for all dynamic laws (6) from D such that $s_1 \models g$ and $s_0 \cup a \models h$.

For any $\mathcal{C}$-description D, the transition system $T(D)$ *represented by* D is defined as follows:

- the vertices of $T(D)$ are the functions from fluents to truth values that satisfy the formulas $g \rightarrow f$ for all static laws (5) from D;
- $\langle s_0, a, s_1\rangle$ is a transition of $T(D)$ iff s_1 is the only function from fluents to truth values that satisfies $D^{\langle s_0, a, s_1\rangle}$.

Discussion The semantics of $\mathcal{C}$ is based on the distinction between what is true and what has a cause. The static law (5) says that there is a cause for f to hold in a state if g is a true assertion about the values of fluents in that state. The dynamic law (6) says that there is a cause for f to hold in the resulting state of a transition if (i) g is a true assertion about the values of fluents in that state, and (ii) h is a true assertion about the values of fluents at the beginning of the transition and about the execution of elementary actions during the transition.

For instance, the dynamic law

$$\textbf{caused } q \textbf{ if } \text{true } \textbf{after } e \wedge p, \tag{7}$$

where p and q are fluents and e is an elementary action, expresses that there is a cause for q to hold in the resulting state of a transition if e is executed during the transition and p holds at the beginning of the transition. (Intuitively, the execution of e is the cause.) The dynamic law

$$\textbf{caused } l \textbf{ if } l \textbf{ after } l, \tag{8}$$

where l is a fluent literal, expresses that there is a cause for l to hold after a transition if l holds both before and

after the transition. (Intuitively, commonsense inertia is the cause.)

The reduct of a $\mathcal{C}$-description D relative to a transition $\langle s_0, a, s_1\rangle$ is the set of all formulas that are caused to hold after this transition, according to D. The condition

$$s_1 \textit{ is the only function from fluents to truth values that satisfies } D^{\langle s_0,a,s_1\rangle}$$

in the definition of the semantics of $\mathcal{C}$ expresses the principle of universal causation (McCain & Turner 1997): whatever is true, has a cause, and the other way around.

Example 2. Consider, for example, the $\mathcal{C}$-description consisting of the dynamic law (7), the static law

$$\textbf{caused } \neg r \textbf{ if } q, \tag{9}$$

and the dynamic laws (8) for the fluent literals

$$p,\ \neg p,\ q,\ \neg q,\ r,\ \neg r \tag{10}$$

as l. It describes the same transition system as the $\mathcal{B}$-description from Example 1. For instance, its reduct relative to the second of transitions (4) consists of three formulas: q (contributed by dynamic law (7)), $\neg r$ (contributed by static law (9)), and p (contributed by dynamic law (8) with p as l). The resulting state $\{p, q, \neg r\}$ of transition (4) is the only function from fluents to truth values that satisfies these formulas.

The Translation

For any $\mathcal{B}$-description D, its *$\mathcal{C}$-image* is the $\mathcal{C}$-description obtained from D by

- rewriting each static law (1) as

$$\textbf{caused } l \textbf{ if } \bigwedge_{l' \in c} l';$$

- rewriting each dynamic law (2) as

$$\textbf{caused } l \textbf{ if } \text{true } \textbf{after } e \wedge \bigwedge_{l' \in c} l';$$

- adding the dynamic laws (8) for all fluent literals l.

For instance, the $\mathcal{C}$-description from Example 2 is the $\mathcal{C}$-image of the $\mathcal{B}$-description from Example 1.

Generally, a $\mathcal{B}$-description and its $\mathcal{C}$-image are not equivalent to each other, that is to say, they may represent different transition systems. But there is a simple syntactic condition guaranteeing the equivalence of a $\mathcal{B}$-description to its $\mathcal{C}$-image. The *fluent dependency graph* of a $\mathcal{B}$-description is the directed graph such that

- its vertices are arbitrary fluent literals, and
- it has an edge from l to l' iff the description contains a static law (1) such that $l' \in c$.

For instance, the fluent dependency graph of the $\mathcal{B}$-description from Example 1 has the vertices (10) and one edge, from $\neg r$ to q.

Soundness Theorem: *A $\mathcal{B}$-description is equivalent to its $\mathcal{C}$-image if its fluent dependency graph is acyclic.*

Without the acyclicity assumption, the assertion of the theorem would be incorrect. Consider, for instance, the $\mathcal{B}$-description

$$\begin{array}{l} p \textbf{ if } q, \\ q \textbf{ if } p. \end{array} \tag{11}$$

There are no elementary actions in the underlying signature, so that the only action here is empty. The corresponding transition system has the states $\{p, q\}$ and $\{\neg p, \neg q\}$ and the transitions

$$\langle\{p, q\}, \emptyset, \{p, q\}\rangle,$$
$$\langle\{\neg p, \neg q\}, \emptyset, \{\neg p, \neg q\}\rangle.$$

The $\mathcal{C}$-image of (11) has one more transition, besides these two:

$$\langle\{\neg p, \neg q\}, \emptyset, \{p, q\}\rangle.$$

Indeed, the reduct of the $\mathcal{C}$-image of (11) relative to this transition consists of the formulas p and q, contributed by the translations

$$\begin{array}{l} \textbf{caused } p \textbf{ if } q, \\ \textbf{caused } q \textbf{ if } p \end{array}$$

of static laws (11).

We will return to the discussion of this counterexample in the conclusion.

Proof of the Soundness Theorem

The proof is based on comparing two logic programs: one that characterizes the transitions of a transition system described in $\mathcal{B}$ (Balduccini & Gelfond 2003),[2] the other providing a similar characterization for a subset of the language $\mathcal{C}$ (Lifschitz & Turner 1999).

The atoms occurring in these logic programs are symbols of the forms

$$f(0),\ f(1),\ e(0),$$

where f is a fluent and e is an elementary action. The programs are nondisjunctive, but they contain classical negation (Gelfond & Lifschitz 1991). So the heads of their rules are these atoms and their classical negations

$$\neg f(0),\ \neg f(1),\ \neg e(0).$$

The body of a rule is a list of such literals, possibly with the negation as failure symbol *not* prepended to some of them.

We will be only interested in *complete* answer sets, that is, in the sets containing exactly one member of each complementary pair $\{f(0), \neg f(0)\}$, $\{f(1), \neg f(1)\}$, $\{e(0), \neg e(0)\}$.

[2] In that paper, $\mathcal{B}$ is referred to as $\mathcal{AL}$.

Translating $\mathcal{B}$-Descriptions into Logic Programming

For any $\mathcal{B}$-description D, its *logic programming representation* $\mathrm{LP}(D)$ is the logic program consisting of

- the rules

$$l(t) \leftarrow l_1(t), \ldots, l_m(t) \qquad (t \in \{0,1\}) \tag{12}$$

for each static law

$$l \textbf{ if } l_1, \ldots, l_m \tag{13}$$

from D;

- the rule

$$l(1) \leftarrow e(0), l_1(0), \ldots, l_n(0) \tag{14}$$

for each dynamic law

$$e \textbf{ causes } l \textbf{ if } l_1, \ldots, l_n \tag{15}$$

from D;

- the rules

$$\begin{array}{l} f(1) \leftarrow f(0), not\ \neg f(1), \\ \neg f(1) \leftarrow \neg f(0), not\ f(1) \end{array} \tag{16}$$

for every fluent f.

The following assertion is a restatement, in different notation, of Lemma 4 from (Balduccini & Gelfond 2003).[3]

Lemma 1 *For any $\mathcal{B}$-description D, any sets s_0, s_1 of fluent literals, and any action a, $\langle s_0, a, s_1\rangle$ is a transition of $T(D)$ iff the program*

$$\mathrm{LP}(D) \cup \{l(0) : l \in s_0 \cup a\} \tag{17}$$

has an answer set X such that $s_1 = \{l : l(1) \in X\}$.

By C we denote the set consisting of the rules

$$\begin{array}{l} f(0) \leftarrow not\ \neg f(0), \\ \neg f(0) \leftarrow not\ f(0) \end{array} \tag{18}$$

for all fluents f, and

$$\begin{array}{l} e(0) \leftarrow not\ \neg e(0), \\ \neg e(0) \leftarrow not\ e(0) \end{array} \tag{19}$$

for all elementary actions e.

[3]The main difference is that the program in that paper does not contain classical negation. Instead of complementary fluent literals $f(t)$, $\neg f(t)$ it uses two different atoms and includes a constraint that does not allow both atoms to belong to the same answer set. Adding such a constraint has the same effect as replacing one of the atoms by the classical negation of the other (Gelfond & Lifschitz 1991, Proposition 2). Furthermore, program (17) contains the facts $\neg e(0)$ for all elementary actions e that do not belong to a; these facts do not correspond to any rules of the program from (Balduccini & Gelfond 2003). The presence of these facts in our program is clearly inessential, because literals of the form $\neg e(0)$ do not occur in the other rules. Finally, the version of $\mathcal{B}$ in (Balduccini & Gelfond 2003) includes an additional construct, **impossible if**, which is not used in this note.

Lemma 2 *For any $\mathcal{B}$-description D, any sets s_0, s_1 of fluent literals, and any action a, $\langle s_0, a, s_1\rangle$ is a transition of $T(D)$ iff the set*

$$\{l(0) : l \in s_0 \cup a\} \cup \{l(1) : l \in s_1\} \tag{20}$$

is an answer set of the program $\mathrm{LP}(D) \cup C$.

Proof By Lemma 1, $\langle s_0, a, s_1\rangle$ is a transition of $T(D)$ iff there exists a set X of literals such that (i) X is an answer set of (17) and (ii) $s_1 = \{l : l(1) \in X\}$. Any set X satisfying conditions (i) and (ii) contains the literals $l(0)$ for all $l \in s_0 \cup a$ (because (17) contains each of these literals as a fact) and the literals $l(1)$ for all $l \in s_1$. In other words, every such X is a superset of (20). Since (20) is complete, it follows that X equals (20). Consequently the assertion of Lemma 1 can be reformulated as follows: $\langle s_0, a, s_1\rangle$ is a transition of $T(D)$ iff (20) is an answer set of (17). It remains to observe that the reduct of (17) with respect to (20) coincides with the reduct of $\mathrm{LP}(D) \cup C$ with respect to (20).

Translating $\mathcal{C}$-Descriptions into Logic Programming

In this section, D is a $\mathcal{C}$-description such that the formula f in each of its static laws (5) and dynamic laws (6) is a literal, and the formulas g and h are conjunctions of literals. The *logic programming representation* $\mathrm{LP}(D)$ of D is the logic program[4] consisting of

- the rules

$$l(t) \leftarrow not\ \overline{l_1(t)}, \ldots, not\ \overline{l_m(t)} \qquad (t \in \{0,1\}) \tag{21}$$

for each static law

$$\textbf{caused } l \textbf{ if } l_1 \wedge \cdots \wedge l_m \tag{22}$$

from D, and

- the rule

$$l(1) \leftarrow not\ \overline{l_1(1)}, \ldots, not\ \overline{l_m(1)}, l_{m+1}(0), \ldots, l_n(0) \tag{23}$$

for each dynamic law

$$\textbf{caused } l \textbf{ if } l_1 \wedge \cdots \wedge l_m \textbf{ after } l_{m+1} \wedge \cdots \wedge l_n \tag{24}$$

from D.

The following assertion is a restatement, in different notation, of Corollary 1 from (Lifschitz & Turner 1999).

Lemma 3 *A complete set X of literals is an answer set of $\mathrm{LP}(D) \cup C$ iff X has the form (20) for some transition $\langle s_0, a, s_1\rangle$ of $T(D)$.*

A *split mapping* for D is a function λ from fluent literals to nonnegative integers[5] such that, for every static law (22) and every dynamic law (24) from D,

$$\lambda(l_1), \ldots, \lambda(l_m) \leq \lambda(l).$$

[4]By $\overline{l}$ we denote the literal complementary to l.

[5]In (Lifschitz & Turner 1999), the values of a split mapping are ordinals. We do not need infinite ordinals here, because the set of fluents is assumed to be finite.

If λ is a split mapping for D then in rules (21) and (23) we can replace any occurrences of *not* $\overline{l_i(t)}$ such that $\lambda(l_i) < \lambda(l)$ with $l_i(t)$ without affecting the complete answer sets of $\mathrm{LP}(D) \cup C$ (Lifschitz & Turner 1999, Proposition 2).

Proof of the Theorem

Let D be a $\mathcal{B}$-description with an acyclic fluent dependency graph, and let D' be the $\mathcal{C}$-image of D. It is clear that the transition systems described by D and by D' have the same states. We need to show that they have the same transitions.

Let s_0 and s_1 be functions from fluents to truth values, let a be an action, and let X be the corresponding set (20). By Lemma 3, $\langle s_0, a, s_1\rangle$ is a transition of $T(D')$ iff X is an answer set of the program $\mathrm{LP}(D') \cup C$. From the definition of the $\mathcal{C}$-image and the definition of the logic programming representation of a $\mathcal{C}$-description we see that program $\mathrm{LP}(D')$ consists of

- rules (21) for each static law (13) from D,
- rule (14) for each dynamic law (15) from D,
- rules (16) for every fluent f.

For any fluent literal l, let $\lambda(l)$ be the length of the longest path in the fluent dependency graph of D that starts at l. Since the graph is finite and acyclic, this is a sound definition. It is clear that λ is a split mapping for D', and that the strict inequality

$$\lambda(l_1), \ldots, \lambda(l_m) < \lambda(l)$$

holds for each static law (22) of D'. According to the concluding remark of the section on translating $\mathcal{C}$-descriptions into logic programming, the complete answer sets of $\mathrm{LP}(D') \cup C$ will not be affected by replacing each pair of rules (21) with rules (12). This replacement turns $\mathrm{LP}(D')$ into $\mathrm{LP}(D)$. Consequently $\langle s_0, a, s_1\rangle$ is a transition of $T(D')$ iff X is an answer set of the program $\mathrm{LP}(D) \cup C$. By Lemma 2, it follows that $T(D)$ and $T(D')$ have the same transitions.

Conclusion

We have shown that $\mathcal{B}$-descriptions with acyclic static dependencies can be easily translated into $\mathcal{C}$. The proof exploits the close relationship of these languages to logic programming under the answer set semantics.

Counterexample (11) shows that the acyclicity assumption is essential. The static laws in that example have a simple intuitive meaning: they can be viewed as saying that p is synonymous with q. Expressing synonymy in $\mathcal{C}$ requires that we use formulas syntactically more complex than literals. For instance, (11) can be translated into $\mathcal{C}$ by the static law

$$\textbf{caused } p \leftrightarrow q \textbf{ if } \mathrm{true}.$$

On the other hand, the static law

$$\textbf{caused } p \textbf{ if } p,$$

syntactically similar to the trivial "inference rule"

$$p \textbf{ if } p,$$

expresses in $\mathcal{C}$ the idea that p holds by default. This can be useful for knowledge representation purposes (see the example of a spring-loaded door in (Giunchiglia & Lifschitz 1998)).

The need to postulate inertia by including dynamic laws (8) makes $\mathcal{C}$ less concise than $\mathcal{B}$. There are fluents, on the other hand, whose values tend to change in a specific way, rather than remain unchanged (see the pendulum example in (Giunchiglia & Lifschitz 1998)); in such cases, the possibility of *not* including the inertia assumption for a fluent may be useful.

The design of each of the languages $\mathcal{B}$ and $\mathcal{C}$ explores a certain approach to the semantics of static causation in the simplest setting. Each of the languages was meant to be extended in several ways. For instance, $\mathcal{C}+$ (Giunchiglia *et al.* 2004) is an extension of $\mathcal{C}$ in which values of a fluent may not be Boolean, and in which a fluent can be defined in terms of other fluents. The language H (Chintabathina, Gelfond, & Watson 2005) expands $\mathcal{B}$ by allowing representation of continuos processes. The language $\mathcal{ALM}$ (Gelfond & Inclezan 2009) is a modular language similar to $\mathcal{B}$, and MAD (Lifschitz & Ren 2006; Ren 2009) is a modular language similar to $\mathcal{C}$. Comparing the expressivity of such extensions will be more interesting than comparing the basic versions reviewed in this note.

Acknowledgements

Many thanks to the anonymous referees for useful comments. The first author was partially supported by the National Science Foundation under Grant IIS-1018031.

References

Balduccini, M., and Gelfond, M. 2003. Diagnostic reasoning with A-Prolog. *Theory and Practice of Logic Programming* 3(4-5):425–461.

Chintabathina, S.; Gelfond, M.; and Watson, R. 2005. Modeling hybrid domains using process description language[6]. In *Proceedings of Workshop on Answer Set Programming: Advances in Theory and Implementation (ASP'05)*.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3–4):189–208.

Gelfond, M., and Inclezan, D. 2009. Yet another modular action language. In *Proceedings of the Second International Workshop on Software Engineering for Answer Set Programming*[7], 64–78.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9:365–385.

[6] `http://ceur-ws.org/vol-142/page303.pdf`

[7] `http://www.sea09.cs.bath.ac.uk/downloads/sea09proceedings.pdf`

Gelfond, M., and Lifschitz, V. 1993. Representing action and change by logic programs. *Journal of Logic Programming* 17:301–322.

Gelfond, M., and Lifschitz, V. 1998. Action languages[8]. *Electronic Transactions on Artificial Intelligence* 3:195–210.

Giunchiglia, E., and Lifschitz, V. 1998. An action language based on causal explanation: Preliminary report. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, 623–630. AAAI Press.

Giunchiglia, E.; Lee, J.; Lifschitz, V.; McCain, N.; and Turner, H. 2004. Nonmonotonic causal theories. *Artificial Intelligence* 153(1–2):49–104.

Lifschitz, V., and Ren, W. 2006. A modular action description language. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, 853–859.

Lifschitz, V., and Turner, H. 1999. Representing transition systems by logic programs. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, 92–106.

McCain, N., and Turner, H. 1997. Causal theories of action and change. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, 460–465.

Pednault, E. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In Brachman, R.; Levesque, H.; and Reiter, R., eds., *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 324–332.

Ren, W. 2009. *A Modular Language for Describing Actions*[9]. Ph.D. Dissertation, University of Texas at Austin.

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

Turner, H. 1998. *Causal Action Theories and Satisfiability Planning*. Ph.D. Dissertation, University of Texas at Austin.

[8] `http://www.ep.liu.se/ea/cis/1998/016/`

[9] `http://www.cs.utexas.edu/users/rww6/dissertation.pdf`