Logics for Action

Michael P. Fourman

School of Informatics
The University of Edinburgh, Scotland, UK
michael.fourman@ed.ac.uk

Abstract. Logics of action, for reasoning about the effects of state change, and logics of belief, accounting for belief revision and update, have much in common. Furthermore, we may undertake an action because we hold a particular belief, and revise our beliefs in the light of observed consequences of an action. So studies of these two aspects are inevitably intertwined. However, we argue, a clear separation of the two is helpful in understanding their interactions.

We give a semantic presentation of such a separation, introducing a semantic setting that supports one logic for describing the effects of actions, which are modeled as changing the values of particular atomic properties, or *fluents*, and another for expressing more complex facts or beliefs about the world. We use a simple *state-logic*, to account for state change, and show how it can be integrated with a variety of *domain-logics*, of fact or belief, for reasoning about the world. State- and domain-logics are linked, syntactically and semantically; but separate.

The state-logic, our *logic for action*, is quantified propositional logic. Bounded existential propositional quantification is used to specify which literals may be established by a given action. This provides a logically transparent account of the treatment of updates introduced in [1], which itself provided a treatment of the frame problem extending the uses in event calculi of the *forget* operator, *occlude* and *release* predicates, all published in 1994 [2–4]. Our, purely classical, logical setting can be seen as a recasting of the state-transition model of action, underlying STRIPS [5] and ADL [6].

Our treatment, like ADL, caters for actions with conditional effects, which can handle ramifications unrepresentable in classic STRIPS; it also includes concurrent actions, non-deterministic effects, and domain axioms or integrity constraints, which are not present in ADL. We introduce an operator, novel in this context, that computes fixpoints of certain *monotone* actions, to handle the problem of nested ramification.

We exploit non-determinacy to provide a decomposition of actions, as compositions of simpler fundamental operations on states. These decompositions provide a basis for the static analysis of interference, which is applied in our treatment of iteration, re-ordering and concurrent composition of actions.

Our setting combines operational and inferential aspects. Domain logics are inferential, and can vary, from classical to exotic. The state logic provides an operational calculus of actions that retains the clarity and simplicity of propositional reasoning, which underlies STRIPS' enduring appeal. It also has practical benefits: applications can exploit propositional reasoning tools, such as BDD packages and SAT solvers to provide a direct route to efficient implementations [7].

keyword action, transition system, logic, semantics, frame problem, representation

1 Introduction

The philosophers have only interpreted the world, in various ways;
the point is to change it. *Marx [10]*

Beliefs represent our attempts to interpret the world; actions change it.

This paper provides a setting that will allow us to give an account of the following simple narrative.

> *Before setting off on a trip, I pack a power adapter for my laptop (because I will need it and don't believe I have one packed already). Packing something ensures, non-deterministically, that it is in either my briefcase or my suitcase. I pack, place myself, briefcase and suitcase in the car, and drive the car from here to there. On arrival, I carry briefcase and suitcase from the car to my hotel room. Unpacking, I find not one, but two adapters, one in the briefcase, one in the suitcase. I realise that my earlier belief was mistaken: I* had *already packed an adapter. I hope my son won't mind that I have them both and he has none!*

We take a semantics-led approach to logic, and start from a simple platonic model of a discrete sequence of static states evolving as the result of a sequence of actions. We use this to provide two different interpretations for two different logics: a dynamic state-logic, and a static domain-logic. This leads to a setting that allows us to give an account of actions featuring *non-determinacy* and *nested ramifications*.

Starting from this formal model of the world, we provide two different interpretations for two different logics: state-logic, and a static domain logic.

We have a set of state variables. On the one hand, these represent states as configurations, or *valuations*—total functions that assign a value to each variable—a value which actions may change. On the other hand, we view each configuration in state-space as encoding a model that provides semantic grounding for our logics of fact and belief, representing a momentarily static view of the world.

The relationship between state-logic and domain logic can also be viewed syntactically: state variables are variables of the state-logic, and they correspond syntactically to (and can, conservatively, be introduced as) constants in the domain-logic.

We conclude this introduction with a brief description of the sections that follow.

In §2 we use quantified propositional logic to provide a setting in which various "update operators" are exposed as simple logical operations.

In §3 we introduce an abstract mathematical notion of action. Semantically, we allow any function from states to sets of states as the effect of an action.[1] The state logic provides syntax for building simple component actions. We build actions generalising those of STRIPS and ADL compositionally, from these primitives.

In §3.4, we extend the conditional effects of ADL by introducing a fixpoint operator, which allows us to handle nested ramification.

[1] We represent these formally by union-preserving total functions on the set of sets of states, since we find it technically and notationally simpler.

§4 describes the interface between state-logic and domain-logic, and touches on the issue of domain constraints.

In §5, we return to assess how well our account addresses the issue of modeling our narrative.

2 State Logic

Our state-logic is just predicate logic with state variables as variables. We show how standard logical operations can be used to address the frame-problem and represent state-change.

Let I be a set of *state-variables*, with each $x \in I$ ranging over a non-empty set $\mathcal{S}_x$ of possible values.[2] We consider states, or valuations, $\mathbf{v}$ that assign a value to each state-variable. So, valuations are elements of the product *state-space*, $\mathcal{S}$:

$$\mathbf{v} \in \mathcal{S} = \prod_{x \in I} \mathcal{S}_x \qquad \mathbf{v} = \langle \mathbf{v}_x \in \mathcal{S}_x \mid x \in I \rangle \tag{1}$$

A *condition*, $\mathcal{C}$, is a subset of the state space, $\mathcal{S}$. We introduce a *logic* whose expressions will denote conditions. For $x, y \in I$, we define the (postfix) substitution operations $[y/x]$ (substitute y for x), on conditions and valuations, semantically, by

$$\mathcal{C}[y/x] = \{\mathbf{v} \mid \mathbf{v}[y/x] \in \mathcal{C}\} \quad \text{where} \begin{cases} (\mathbf{v}[y/x])_x = \mathbf{v}_y \\ (\mathbf{v}[y/x])_z = \mathbf{v}_z \quad \text{for } z \text{ different from } x \end{cases}$$

Definition 1 (Syntax). *Expressions are built, as usual, from atoms. Each condition $\mathcal{C}$ is an* atomic *expression.*[3] *If φ and ψ are expressions, x is a state-variable, and α is an action, then the following are expressions:*

$$\varphi \wedge \psi, \;\; \varphi \vee \psi, \;\; \neg\varphi, \;\; \varphi \to \psi, \;\; \varphi \leftrightarrow \psi, \;\; \exists x.\, \varphi, \;\; \forall x.\, \varphi, \;\; \alpha.\varphi \tag{2}$$

We write $\top, \bot$ for the maximal ($\mathcal{S}$) and minimal ($\emptyset$) conditions; $\bar{x}$ for a set of variables; $\bar{x}, \bar{y}$, for the union of sets of variables; and $\exists \bar{x}.\, \psi$, and $\forall \bar{x}.\, \psi$ for quantification over several variables.

---

[2] We also allow ourselves, for convenience, a supply of auxiliary variables. These are innocuous, once quantified—which will be done wherever they are introduced. We use Greek letters μ, ν, to denote such a variable, which will always be **new**, in the sense that it occurs only where explicitly shown.

[3] We adopt standard mathematical notations (eg. $x = y$ for $\{\mathbf{v} \mid \mathbf{v}_x = \mathbf{v}_y\}$) as shorthand for common conditions: where $x \in I$ and $a \in \mathcal{S}_x$, we write $x = a$ for $\{\mathbf{v} \mid \mathbf{v}_x = a\}$; where b, x, y are boolean-valued variables, write b for $\{v \mid v_b = \texttt{true}\}$, and $x \leq y$ for $\{\mathbf{v} \mid \mathbf{v}_x \leq \mathbf{v}_y\}$ (where booleans have the usual order; $\bot \leq \top$); similarly $\geq$.

Definition 2 (Semantics). *The* interpretation *of a static expression φ is the condition $[\![\varphi]\!]$* [4] *determined by the following inductive rules:*

$$[\![\mathcal{C}]\!] = \mathcal{C} \qquad [\![\alpha.\varphi]\!] = [\![\alpha]\!][\![\varphi]\!] \qquad [\![\neg\varphi]\!] = \mathcal{S} \setminus [\![\varphi]\!] \tag{3}$$

$$[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!] \qquad [\![\varphi \vee \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!] \tag{4}$$

$$[\![\varphi \to \psi]\!] = [\![\neg\varphi \vee \psi]\!] \qquad [\![\varphi \leftrightarrow \psi]\!] = [\![\varphi \to \psi \wedge \psi \to \varphi]\!] \tag{5}$$

$$\mathbf{w} \in [\![\exists x.\, \varphi]\!] \quad \textbf{\textit{iff}} \quad \exists \mathbf{v} \in [\![\varphi]\!].\, \forall y \neq x.\, \mathbf{v}_y = \mathbf{w}_y \tag{6}$$

$$\mathbf{w} \in [\![\forall x.\, \varphi]\!] \quad \textbf{\textit{iff}} \quad \forall \mathbf{v}.\, (\forall y \neq x.\, \mathbf{v}_y = \mathbf{w}_y) \to \mathbf{v} \in [\![\varphi]\!] \tag{7}$$

For any correct, syntactic definition of substitution, $[\![\varphi[y/x]]\!] = [\![\varphi]\!][y/x]$.

$$\text{We say,} \quad \textit{“}\varphi \textit{ is equivalent to } \psi\textit{”} \quad \varphi \equiv \psi \quad \textbf{iff} \quad [\![\varphi]\!] = [\![\psi]\!] \tag{8}$$

$$\textit{“}\varphi \textit{ entails } \psi\textit{”} \quad \varphi \vDash \psi \quad \textbf{iff} \quad [\![\varphi]\!] \subseteq [\![\psi]\!] \tag{9}$$

$$\text{Note that,} \quad b \equiv b \;=\; \texttt{true} \qquad \neg b \equiv b \;=\; \texttt{false} \tag{10}$$

If every state-variable is boolean-valued, then the fragment of this logic consisting of expressions without conditions amounts to the standard interpretation of Quantified Boolean Formulæ (QBF) over the state-variables. If the domain of each state-variable is finite, the logic is known as Quantified Propositional Logic. This finiteness restriction is introduced later in this paper, when we come to compute fixpoints of actions.

We remark that our state-logic and semantics can be viewed as a presentation of the classical (Tarski) satisfaction relation ($\vDash_T$) for many-sorted first-order predicate logic (FOL) with non-empty sorts and without function symbols: we take as state-variables just the variables of the logic. To establish this correspondence, we introduce a condition, $[\![P(\bar{x})]\!]$, for each atomic formula, $P(\bar{x})$, of FOL, where $(\bar{x})$ represents all the free variables of the atom: $[\![P(\bar{x})]\!] = \{\mathbf{v} \mid \mathbf{v} \vDash_T P(\bar{x})\}$ It is then straightforward to verify, by induction on the structure of formulae, that $[\![\varphi]\!] = \{\mathbf{v} \mid \mathbf{v} \vDash_T \varphi\}$. Thus, in addition to the classical propositional identities, all the identities and entailments of classical first-order predicate logic are valid, in particular the following:

$$\exists \bar{x}.\, (\varphi \vee \psi) \equiv \exists \bar{x}.\, \varphi \vee \exists \bar{x}.\, \psi \qquad \exists \bar{x}.\, (\varphi \wedge \psi) \vDash \exists \bar{x}.\, \varphi \wedge \exists \bar{x}.\, \psi \tag{11}$$

$$\forall \bar{x}.\, \varphi \wedge \forall \bar{x}.\, \psi \equiv \forall \bar{x}.\, (\varphi \wedge \psi) \qquad \forall \bar{x}.\, \varphi \vee \forall \bar{x}.\, \psi \vDash \forall \bar{x}.\, (\varphi \vee \psi) \tag{12}$$

We will apply this setting to the representation of actions by taking fluents as state-variables, so that a state is an assignment of values to the fluents.

Remark 1 (Dependence and Independence). For any condition, $\mathcal{C}$, and variables, $\bar{x}$,

$$\mathcal{C} \equiv \exists \bar{x}.\, \mathcal{C} \quad \textbf{iff} \quad \mathcal{C} \equiv \forall \bar{x}.\, \mathcal{C} \quad \textbf{iff} \quad \exists \bar{x}.\, \mathcal{C} \equiv \forall \bar{x}.\, \mathcal{C}$$

[4] We also say $[\![\varphi]\!]$ *describes* φ. In the literature, $[\![\varphi]\!]$ is often denoted *Mod*(φ).

If $\bar{x}$ is a set of variables, $\mathcal{C}$ a condition, and φ an expression, there is an obvious semantic notion of independence[5]:

$$\mathcal{C} \bowtie \bar{x} \quad \textbf{iff} \quad \mathcal{C} \equiv \forall \bar{x}.\mathcal{C} \qquad \mathcal{C} \text{ is } \textit{independent of } \bar{x} \tag{13}$$

$$\varphi \bowtie \bar{x} \quad \textbf{iff} \quad [\![\varphi]\!] \bowtie \bar{x} \qquad \varphi \text{ is } \textit{independent of } \bar{x} \tag{14}$$

$$\mathcal{C} \rhd x \quad \textbf{iff} \quad \textbf{not } \mathcal{C} \bowtie x \qquad \mathcal{C} \textit{is dependent on } x \tag{15}$$

$$\varphi \rhd x \quad \textbf{iff} \quad \textbf{not } \varphi \bowtie x \qquad \varphi \textit{ is dependent on } x \tag{16}$$

Definition 3. *We use the usual notion of* free variables*, $FV(\varphi) \subseteq I$, of a formula φ. Free variables are introduced by the atomic cases $FV(\mathcal{C}) = \{x \mid \mathcal{C} \rhd x\}$; quantifiers bind variables $FV(\exists x.\,\varphi) = FV(\forall x.\,\varphi) = FV(\varphi) \setminus \{x\}$; other connectives cumulate free variables $FV(\varphi \wedge \psi) = FV(\varphi \vee \psi) = FV(\varphi) \cup FV(\psi)$.*

Remark 2. If $x \notin FV(\varphi)$ then $\varphi \bowtie x$.

$$\text{If } \varphi \bowtie \bar{x} \text{ then } \varphi \wedge \exists \bar{x}.\,\psi \equiv \exists \bar{x}.\,(\varphi \wedge \psi) \text{ and } \forall \bar{x}.\,(\varphi \vee \psi) \equiv \varphi \vee \forall \bar{x}.\,\psi$$

Proof. By definition, the first claim holds for atomic formulæ; the general case follows by induction on the structure of φ. For the second, we have to prove the converse of (11). If $\mathbf{v} \in [\![\varphi \wedge \exists \bar{x}.\,\psi]\!]$ then there is some $\mathbf{w} \in [\![\psi]\!]$ that agrees with $\mathbf{v}$ except on $\bar{x}$—but then $\mathbf{w} \in [\![\exists \bar{x}.\,\varphi]\!] = [\![\varphi]\!]$, so $\mathbf{w} \in [\![\varphi \wedge \psi]\!]$, and thus $\mathbf{v} \in [\![\exists \bar{x}.\,(\varphi \wedge \psi)]\!]$. The argument for the third is similar.

Independence is important in establishing non-interference between actions. We rely on this, and remark 3 below, to provide syntactic control of interference.

Definition 4. *We define $\varphi \curlywedge l$ (φ is monotone in l), for l a literal:*

$$\varphi \textit{ is monotone in } x, \quad \textbf{\textit{iff}} \quad \exists \nu.\,(\nu \leq x \wedge \varphi[\nu/x]) \vDash \varphi \tag{17}$$

$$\varphi \textit{ is monotone in } \neg x, \quad \textbf{\textit{iff}} \quad \exists \nu.\,(\nu \geq x \wedge \varphi[\nu/x]) \vDash \varphi \tag{18}$$

Observe that entailments in the reverse direction are trivial.[6]
We use l, m for literals, and say $\neg x$ (resp. x) *is true in* $\mathbf{v}$ if $\mathbf{v}_x = \bot$ (resp. $\mathbf{v}_x = \top$). We mildly abuse notation by writing $\mathbf{v}_l$ to stand for $\mathbf{v}_x$ if l is x, and $\neg \mathbf{v}_x$ if l is $\neg x$.

Remark 3. A standard inductive argument shows that if x occurs only positively (resp. negatively) in φ, then φ is monotone in x (resp. $\neg x$). Conversely, if φ is monotone in l, observe that any occurrence of $\neg l$ in a conjunctive normal form for φ is redundant, and can be deleted. So φ is monotone in l iff there is a CNF for φ in which $\neg l$ does not occur. Observe also that, $\exists \nu \leq x.\,\varphi[\nu/x]$ is monotone in x; dually, $\exists \nu \geq x.\,\varphi[\nu/x]$ is monotone in $\neg x$; furthermore,

$$\exists x\,\varphi \equiv \exists \mu \leq x.\,\exists \nu \geq \mu.\,\varphi[\nu/x] \equiv \exists \mu \geq x.\,\exists \nu \leq \mu.\,\varphi[\nu/x] \tag{19}$$

[5] Part of logical folklore: explicit in Lang [11], but already implicit in [4], for example.
[6] Hereinafter, we use standard abbreviations, such as $\exists \nu \leq x.\,\varphi[\nu/x]$, for bounded quantification.

Lang and Marquis [12, 11] define a boolean function to be *Lit-independent* of a literal l iff it can be expressed in conjunctive normal form without occurrence of l. It is straightforward (this a direct consequence of remark 3) to see that,

$$\varphi \textit{ is monotone in } l \textit{ iff } \varphi \textit{ is Lit-independent of } \neg l$$

If φ describes a set of states, then $\exists x.\varphi$ represents the set of states that are like some state in $[\![\varphi]\!]$ except that they may have a different value for the fluent x:

$$[\![\exists x.\, \varphi]\!] = \{\mathbf{w} \mid \text{for some } \mathbf{v} \in [\![\varphi]\!]. \text{ for all } y, \text{ other than } x,\ \mathbf{v}_y = \mathbf{w}_y\}$$

The bounded quantifier gives finer control—it allows the value of a fluent to change in one direction only. For example, upwards:

$$[\![\exists \nu \leq x.\, \varphi[\nu/x]]\!] = \{\mathbf{w} \mid \exists \mathbf{v} \in [\![\varphi]\!].\mathbf{v}_x \leq \mathbf{w}_x, \text{ and, for all } y \text{ other than } x,\ \mathbf{v}_y = \mathbf{w}_y\}$$

3 Actions

Definition 5. *An* action *is a union-preserving*[7] *total function on the powerset of* $\mathcal{S}$.

$$\alpha : \wp(\mathcal{S}) \to \wp(\mathcal{S})$$

Each action, α, is determined by its values on singletons. We will often abuse notation by writing $\alpha(\mathbf{v})$ for $\alpha(\{\mathbf{v}\})$.

We choose to work with total functions on sets of states, rather than relations, or partial functions, since these sets of states are the semantic counterparts of logical expressions, and we reason about actions, plans and beliefs, by manipulating such sets. Applications can exploit technologies for propositional reasoning, such as BDD packages and SAT solvers to implement these manipulations (cf. [7, 13, 14]). This presentation also makes sequential composition of actions simply composition of functions.

3.1 Primitive Actions

We introduce notation,[8] in the style (**`keyword`** *parameters*), for some primitive actions.[9] The reader is invited to observe that these are indeed actions—that the property of preserving unions obtains.

$$(\mathtt{require}\,\psi)(\mathcal{C}) = [\![\psi]\!] \cap \mathcal{C} \qquad \mathtt{null} = \mathtt{require}\,\top \tag{20}$$

The action $(\mathtt{require}\,\psi)$ is the identity, **`null`**, restricted to states for which ψ holds.

$$(\mathtt{allow}\,x)(\mathcal{C}) = [\![\exists \nu \leq x.\, \mathcal{C}[\nu/x]]\!] \quad (\mathtt{allow}\,\neg x)(\mathcal{C}) = [\![\exists \nu \geq x.\, \wedge\, \mathcal{C}[\nu/x]]\!] \tag{21}$$

[7] Note that, in particular, $\alpha(\emptyset) = \emptyset$, for any action, α.

[8] In these examples, ψ is a fixed expression, x is a state-variable, and l a literal.

[9] Many of these occur in the literature; for example, $(\mathtt{allow}\,l)\varphi$ corresponds to *ForgetLit*$(\varphi, \neg l)$ in [12, 11], and $(\mathtt{change}\,x)\varphi$ to *ForgetVar*(φ, x).

We write $\mathbf{change}\, x$ for $(\mathbf{allow}\, x \ ; \ \mathbf{allow}\, x)$, so $(\mathbf{change}\, x)(\mathcal{C}) = [\![\exists x.\, \mathcal{C}]\!]$. The actions $\mathbf{change}\, x$ and $\mathbf{allow}\, l$ are crucial to our account. We use them as **the** mechanism for state-change, and this is how we address the frame problem.

The action $(\mathbf{when}\ \psi\ \alpha)$ (where ψ is an expression and α an action) is defined by its action on singletons: $(\mathbf{when}\ \psi\ \alpha)(\mathbf{v}) =$ **if** $\mathbf{v} \in [\![\psi]\!]$ **then** $\alpha(\mathbf{v})$ **else** $\{\mathbf{v}\}$. Extending $(\mathbf{when}\ \psi\ \alpha)$ to general conditions, $\mathcal{C}$, gives:

$$(\mathbf{when}\ \psi\ \alpha)(\mathcal{C}) = (\mathcal{C} \cap [\![\neg\psi]\!]) \cup (\alpha)(\mathcal{C} \cap [\![\psi]\!]) \tag{22}$$

Definition 6. *An action α is* deterministic *iff $\forall \mathcal{C}.\, |\alpha(\mathcal{C})| \leq |\mathcal{C}|$; equivalently, iff α is at most single-valued on singletons: $\forall \mathbf{v}.\, |\alpha(\mathbf{v})| \leq 1$. A deterministic action corresponds to a partial function on $\mathcal{S}$.*

Composing Actions Actions can be composed sequentially[10] by function composition, and non-deterministically by set union:

$$(\alpha\ ;\ \beta)(\mathcal{C}) = \beta(\alpha(\mathcal{C})) \qquad\qquad (\alpha \mid \beta)(\mathcal{C}) = \alpha(\mathcal{C}) \cup \beta(\mathcal{C}) \tag{23}$$

Lemma 1 (Easy Identities). [11]

$$\mathbf{allow}\, l; \mathbf{allow}\, l = \mathbf{allow}\, l \tag{24}$$

$$\mathbf{allow}\, l\ ;\ \mathbf{allow}\, m = \mathbf{allow}\, m\ ;\ \mathbf{allow}\, l \tag{25}$$

$$\mathbf{require}\, \varphi\ ;\ \mathbf{require}\, \psi = \mathbf{require}(\varphi \wedge \psi) \tag{26}$$

$$\mathbf{allow}\, x\ ;\ \mathbf{allow}\, \neg x = \mathbf{change}\, x \tag{27}$$

$$\mathbf{allow}\, \bar{l}\ ;\ \mathbf{require}\, \varphi\ ;\ \mathbf{allow}\, \bar{l}\ ;\ \mathbf{require}\, \varphi = \mathbf{allow}\, \bar{l}\ ;\ \mathbf{require}\, \varphi \tag{28}$$

$$\mathbf{require}\, \varphi\ ;\ \mathbf{allow}\, \bar{l}\ ;\ \mathbf{require}\, \varphi\ ;\ \mathbf{allow}\, \bar{l} = \mathbf{require}\, \varphi\ ;\ \mathbf{allow}\, \bar{l} \tag{29}$$

$$\mathbf{when}\, \psi\ (\mathbf{when}\, \varphi\ \alpha) = \mathbf{when}\ (\psi \wedge \varphi)\ \alpha \tag{30}$$

$$\mathbf{when}\, \psi\ (\mathbf{require}\, \psi; \alpha) = \mathbf{when}\, \psi\ (\alpha) \tag{31}$$

If $\varphi \bowtie x$, then

$$\mathbf{allow}\, x\ ;\ \mathbf{require}\, \varphi = \mathbf{require}\, \varphi\ ;\ \mathbf{allow}\, x$$
$$\mathbf{allow}\, \neg x\ ;\ \mathbf{require}\, \varphi = \mathbf{require}\, \varphi\ ;\ \mathbf{allow}\, \neg x \tag{32}$$

If φ is monotone in l, then

$$\mathbf{allow}\, l\ ;\ \mathbf{require}\, \varphi\ ;\ \mathbf{allow}\, l = \mathbf{allow}\, l\ ;\ \mathbf{require}\, \varphi \tag{33}$$

Definition 7 (Reciprocal Actions). *If α is an action, we define the* reciprocal *action*[12] *$\langle\alpha\rangle$:*

$$\langle\alpha\rangle(\mathcal{C}) = \{\mathbf{v} \mid \exists \mathbf{w} \in \alpha(\mathbf{v}).\, \mathbf{w} \in \mathcal{C}\} \tag{34}$$

[10] Note that, for this notion of composition, if either action produces an empty set of results then so does the composite.

[11] We write $\mathbf{allow}\ l, m$ for the composite action $(\mathbf{allow}\ l\ ;\ \mathbf{allow}\ m)$, and, similarly, since order and repetition are irrelevant, $\mathbf{change}\ \bar{x}$, for permitting changes to a set, $\bar{x}$, of state-variables, $\mathbf{allow}\ \bar{l}$ for allowing a set $\bar{l}$ of literals to be established.

[12] Some do not like to call this an action, as many real-world actions cannot simply be reversed. However, from a formal point of view, it has the required form. Irreversibility is more a matter of thermodynamics than logic.

This gives the set of states from which we *may* reach $\mathcal{C}$ by α. An agent performing α then observing $\mathcal{O}$ can infer that $\langle\alpha\rangle\mathcal{O}$ must have held before the performance.

Lemma 2 (Reciprocal Actions).

$$\langle \mathtt{null} \rangle = \mathtt{null} \tag{35}$$

$$\langle \mathtt{change}\, x \rangle = \mathtt{change}\, x \tag{36}$$

$$\langle (\mathtt{allow}\, l) \rangle = (\mathtt{allow}\, \neg l) \tag{37}$$

$$\langle (\mathtt{require}\, \psi) \rangle = (\mathtt{require}\, \psi) \tag{38}$$

$$\langle (\alpha\ ;\ \beta) \rangle = (\langle\beta\rangle\ ;\ \langle\alpha\rangle) \tag{39}$$

Proof. Recall that each action α corresponds to a relation; the reciprocal action $\langle\alpha\rangle$ corresponds to the reciprocal relation.

Remark 4. The related operation[13] $[\alpha]$, which gives the set of states from which we *must* reach $\mathcal{C}$ if we perform α, preserves unions, and thus is an action, iff α is deterministic

$$[\alpha](\mathcal{C}) = \{\mathbf{v} \mid \forall \mathbf{w} \in \alpha(\mathbf{v}).\, \mathbf{w} \in \mathcal{C}\}$$

3.2 STRIPS Actions

Remark 5. In *classic* STRIPS [5] a state is represented by the set of atoms true in that state. Actions are presented by giving three sets of atoms: *preconditions*, ***pre***, that have to be true for the action to be performed, and *effects*, ***delete*** (contained in ***pre***) and ***add*** (disjoint from ***pre***), that specify changes, wrought by the action, to the set of true atoms.

The action $(\boldsymbol{pre}, \boldsymbol{add}, \boldsymbol{delete})$ in classic STRIPS corresponds to the action:

$$\Big(\mathtt{require} \bigwedge_{p \in \boldsymbol{pre}} p\ ;\ \mathtt{allow}\ (\boldsymbol{add} \cup \{\neg d \mid d \in \boldsymbol{delete}\})\ ;\ \mathtt{require} \bigwedge_{a \in \boldsymbol{add}} a \wedge \bigwedge_{d \in \boldsymbol{delete}} \neg d\Big)$$

Definition 8. *The* pre-condition, $\boldsymbol{pre}_\alpha$, *of an action α, is $\langle\alpha\rangle(\mathcal{S})$.*

The set, $\boldsymbol{changes}_\alpha$, of literals that may be set by α *is the smallest*[14] *set $\bar{l}$, of literals, such that, for all $\mathcal{C} \subseteq \mathcal{S}$*

$$\alpha(\mathcal{C}) \subseteq (\mathtt{allow}\, \bar{l})(\mathcal{C}). \tag{40}$$

The post-condition, $\boldsymbol{post}_\alpha$, *of α, is $\alpha(\mathcal{S})$.*

In the case of a classic STRIPS action, α, these operations simply recover its constituent parts: $\alpha = (\mathtt{require}\, \boldsymbol{pre}_\alpha\ ;\ \mathtt{allow}\, \boldsymbol{changes}_\alpha\ ;\ \mathtt{require}\, \boldsymbol{post}_\alpha)$.

Definition 9. *A (generalised)* STRIPS *action σ has the form:*

$$\sigma = (\mathtt{require}\ \boldsymbol{pre}\ ;\ \mathtt{allow}\ \boldsymbol{lits}\ ;\ \mathtt{require}\ \boldsymbol{post})$$

*where **pre** is a formula, **lits** is a set of literals that may be set by the action, and **post** is a formula.*

---

[13] This operator, $[\alpha]$ is the *tidy regression operator* ([6] definition 5.2) for α.

[14] Since α preserves unions, it is sufficient to check (40) for singletons $\mathcal{C} = \{\mathbf{v}\}$, where $\mathbf{v} \in \boldsymbol{pre}_\alpha$. For each $\mathbf{v}$, and set $\bar{l}$ of literals, observe that $\alpha(\mathbf{v}) \subseteq (\mathtt{allow}\, \bar{l})\{\mathbf{v}\}$ iff $\bar{l}$ contains all literals, m, true in some state $\mathbf{w} \in \alpha(\mathbf{v})$, but not in $\mathbf{v}$.

Note that, while classic STRIPS actions are deterministic, generalised STRIPS actions may be non-deterministic. The simple form of our generalised STRIPS actions allows us to give simple explicit descriptions of reciprocal actions and concurrent composition.

Remark 6. Lemma 2 allows us to give the reciprocal of a STRIPS action explicitly:[15]

$$\langle(\texttt{require}\ \textit{pre}\ ;\ \texttt{allow}\ \textit{lits}\ ;\ \texttt{require}\ \textit{post})\rangle \\ = (\texttt{require}\ \textit{post}\ ;\ \texttt{allow}\ \neg\textit{lits}\ ;\ \texttt{require}\ \textit{pre}) \quad (41)$$

Definition 10. *Concurrent Actions We say actions α and β are* independent*, $\alpha \bowtie \beta$, iff $(\alpha\ ;\ \beta) = (\beta\ ;\ \alpha)$, in which case we write $(\alpha\ ||\ \beta)$ for this* concurrent composition*.*

Some pairs of actions cannot meaningfully be performed concurrently: they may have conflicting effects, or one may establish—or destroy—the pre-conditions of the other.

Lemma 3 (Concurrent STRIPS Actions). *Let α and β be generalised* STRIPS *actions:*

$$\alpha = (\texttt{require}\,\textit{pre}_\alpha\ ;\ \texttt{allow}\,\bar{l}_\alpha\ ;\ \texttt{require}\,\textit{post}_\alpha) \\ \beta = (\texttt{require}\,\textit{pre}_\beta\ ;\ \texttt{allow}\,\bar{l}_\beta\ ;\ \texttt{require}\,\textit{post}_\beta)$$

such that the pre-conditions of each are independent of the changes of the other, and the post-conditions of each are monotone in the changes of the other (non-interference):

$$\textit{pre}_\beta \bowtie \bar{l}_\alpha \ \curlywedge\ \textit{post}_\beta \qquad \textit{pre}_\alpha \bowtie \bar{l}_\beta \ \curlywedge\ \textit{post}_\alpha$$

then $\alpha \bowtie \beta$, and $(\alpha\,||\,\beta)$ is given by:

$$(\texttt{require}\ \textit{pre}_\alpha \wedge \textit{pre}_\beta\,;\texttt{allow}\ \bar{l}_\alpha, \bar{l}_\beta\ ;\ \texttt{require}\ \textit{post}_\alpha \wedge \textit{post}_\beta) \quad (42)$$

This form of concurrency has restricted application: the preconditions of both constituent actions must be satisfied as precondition. In the next section we consider conditional effects, which represent a form of *opportunistic* concurrent cooperation.

3.3 ADL actions

Pednault [6] generalises STRIPS by introducing conditional effects. We introduce generalised ADL actions, then discuss how Pednault's actions compare with our treatment.

An archetypical example of the behaviour we want to represent is the briefcase problem. The action of moving the briefcase from home to work should also move the contents of the briefcase. So the location of the laptop should change if it is in the briefcase, but otherwise be inert.

The key idea introduced by Pednault is to allow the set of fluents changed by an action to be context-dependent. Actions of the form $\texttt{when}\,\varphi(\texttt{allow}\,\bar{l};\texttt{require}\,\psi)$ express the conditional effect of allowing changes to $\bar{l}$ in order to require ψ, when the condition φ holds.

[15] Here, $\neg\,\textit{lits}$ denotes the result of negating each of the literals.

Lemma 4. *Given actions γ, δ of form*

$$\gamma = \mathbf{when}\ \varphi_\gamma(\mathbf{change}\ \bar{l_\gamma}; \mathbf{require}\ \psi_\gamma) \tag{43}$$

$$\delta = \mathbf{when}\ \varphi_\delta(\mathbf{change}\ \bar{l_\delta}; \mathbf{require}\ \psi_\delta) \tag{44}$$

such that the conditions and effects of each are monotone in changes made by the other (non-interference):

$$\varphi_\delta \curlywedge \bar{l}_\gamma \curlywedge \psi_\delta \qquad \varphi_\gamma \curlywedge \bar{l}_\delta \curlywedge \psi_\gamma \tag{45}$$

then $\gamma \bowtie \delta$, so $\gamma||\delta$ is well-defined.[16]

Definition 11. *A* (generalised) ADL *action* $(\mathbf{require}\ P; ||_{i\in I}\kappa_i)$ *is formed from a precondition, P followed by the parallel composition of a finite set of conditional effects, $\kappa_i = (\mathbf{when}\ \boldsymbol{cond}_i\ (\mathbf{change}\ \boldsymbol{lits}_i\ ;\ \mathbf{require}\ \boldsymbol{eff}_i))$, where the κ_i are mutually independent.*[17]

Typically, one of the conditions $\boldsymbol{cond}_i$ is $\top$, and this component is viewed as a standard STRIPS action, to which the others are joined as conditional effects. Our treatment of conditional effects is, strictly speaking, incomparable with Pednault's ADL: his definition permits interference between conditional effects; ours allows non-deterministic effects. We have not encountered instances that exploit interfering conditional effects.

Conditional effects were introduced to handle ramification—if the briefcase moves to the station, and the envelope is in the briefcase, then the envelope should move to the station. The solution is fine, as far as it goes. But what if a letter is in the envelope, and the briefcase is in a car, which drives to the station. We want **in**-ness to be transitive, but transitive closure cannot be expressed directly in our propositional setting–nor even in first-order logic. In the following section we introduce a solution to this problem.

3.4 Iterated Actions

In this section we introduce a novel operation that will allow us to cascade conditional effects.

Definition 12. *Let **lits** be a set of literals. We define the ordering $\leq_{\boldsymbol{lits}}$ on states*

$$\mathbf{v} \leq_{\boldsymbol{lits}} \mathbf{w} \quad \textbf{\textit{iff}} \quad \mathbf{w} \in (\mathbf{allow}\,\boldsymbol{lits})\{\mathbf{v}\} \tag{46}$$

For $\mathcal{V}, \mathcal{W}$ sets of states, we say

$$\begin{aligned} \mathcal{V} \leq_{\boldsymbol{lits}} \mathcal{W} \quad \textbf{\textit{iff}} \quad & \textit{for each } \mathbf{v} \in \mathcal{V} \textit{ there is a } \mathbf{w} \in \mathcal{W} \textit{ with } \mathbf{v} \leq_{\boldsymbol{lits}} \mathbf{w}, \textit{ and} \\ & \textit{for each } \mathbf{w} \in \mathcal{W} \textit{ there is a } \mathbf{v} \in \mathcal{V} \textit{ with } \mathbf{v} \leq_{\boldsymbol{lits}} \mathbf{w}. \end{aligned} \tag{47}$$

An action α is monotone *iff, whenever $\mathbf{v} \leq_{\boldsymbol{changes}_\alpha} \mathbf{w}$, then $\alpha\mathbf{v} \leq_{\boldsymbol{changes}_\alpha} \alpha\mathbf{w}$.*

[16] In contrast with the situation in Lemma 3 there is no simple form for this parallel composition, as—and this is the point of conditional effects—the set of variables changed is not uniform across states.

[17] In practice, mutual independence is established statically, through syntactic analysis of monotonicity, applying lemma 4 and remark 3.

Iterating a monotone operator with finite action leads us to a fixed point. We define, as usual, $\alpha^0 = \mathtt{null}, \alpha^{k+1} = (\alpha^k \ ; \ \alpha)$. *The action* $\alpha^\dagger$*, of* iteration to fixpoint *of a monotone action,* α*, is given by* $\alpha^\dagger(\mathbf{v}) = \alpha^n(\mathbf{v})$ *where,* n *is the smallest number such that* $\alpha^{n+1}(\mathbf{v}) = \alpha^n(\mathbf{v})$ *(which exists, thanks to monotonicity).*

Definition 13. *An* extended ADL *action*

$$\left(\mathtt{require}\, P; (||_{i \in I} \kappa_i)^\dagger \right)$$

is formed from a precondition, P *followed by the parallel composition of a finite set of conditional effects,*

$$\kappa_i = (\mathtt{when}\ \boldsymbol{cond}_i\ (\mathtt{change}\ \boldsymbol{lits}_i\ ;\ \mathtt{require}\ \boldsymbol{eff}_i)) \tag{48}$$

where $\boldsymbol{cond}_i$ *and* $\boldsymbol{eff}_i$ *are monotone in all literals* $\boldsymbol{lits}_j$ *(including the case where* $i = j$*).*

The monotonicity conditions suffice to ensure that the κ_i are mutually independent, and $(||_{i \in I} \kappa_i)$ is monotone.

4 Domain Logic

An abstract state-space is of little use, until we connect fluents with meanings. The usual way to make this connection, common to the various approaches in the literature, is to make the fluents represent the values of atomic sentences (ground atoms) of some logic. We do the same.

We use first-order logic as an example of a domain logic; a state of the world corresponds to a first-order structure, or model. To connect with the state logic described above, we illustrate how a state-space is derived from some family of first-order structures. A similar procedure could be used to connect our state-logic to a variety of other logics.

Let $\mathcal{L}$ be a first-order language, without function symbols. Let $\mathcal{X}$ be a finite set. We consider structures for $\mathcal{L}$ with $\mathcal{X}$ as underlying set, and encode these structures as states of a state-space. Recall that such a structure is given by providing an interpretation—a relation on $\mathcal{X}$, of appropriate arity—for each relation symbol of $\mathcal{L}$. We add to $\mathcal{L}$ constant symbols for the elements of $\mathcal{X}$ and take as boolean state-variables the atomic sentences formed by applying relation symbols to these constants. Each state, $\mathbf{v}$, thus assigns a truth value to each of these atomic sentences; these truth values determine the interpretations of the relation symbols in a corresponding structure, $M(\mathbf{v})$.

The fluents, or state-variables, are atomic sentences, or ground atoms of the domain logic. Note that each sentence φ of the domain logic determines a constraint: $\{\mathbf{v} \mid M(\mathbf{v}) \vDash \varphi\}$. We (gently) abuse notation by writing the formula itself in place of this constraint, and thus allow sentences of the domain logic to occur as formulæ of the state logic. An atomic sentence of the domain logic thus occurs as an atomic formula of the state logic both as a constraint and as a boolean state-variable. These two are given identical semantics by equation (3). Similarly, a boolean combination of sentences generates a constraint semantically equivalent to the same boolean combination of the constraints generated by the constituent sentences, so this notational abuse is innocuous.

Note that the domain logic, with variables and quantifiers ranging over elements of $\mathcal{X}$ is quite different from the state logic, whose variables are the fluents. However, the two logics interact syntactically and semantically. Syntactically, the variables and atomic formulæ of the state logic represent sentences of the domain logic. Semantically, each sentence of the domain logic is characterised, up to equivalence for the family of models encoded, by the constraint associated to it above. Taking this constraint as the semantics for a domain logic sentence, we see that where the two logics meet syntactically, they agree semantically.

We can extend this approach to a language with function symbols. We extend the example above to add a single, binary function symbol, f, leaving the reader to generalise. For each pair, a, b, of elements of $\mathcal{X}$, we add a fluent, or state-variable, named $f(a, b)$, whose value ranges over elements of $\mathcal{X}$. In the domain logic, the values of the various fluents $f(a, b)$ where $a, b \in \mathcal{X}$ determine the interpretation of f in the corresponding structure. In the state logic, the choice of name for the fluent ensures semantic and syntactic alignment for the atomic formulæ $f(a, b) = c$.

Lemma 5. *Let φ and A be sentences of the domain logic, where A is a ground atom. Then the constraint corresponding to φ is independent of the fluent corresponding to A unless A is a substitution instance of some atomic formula occurring in φ. Furthermore, if A arises only as a substitution instance of atomic formulæ that occur positively in φ, then the constraint will be monotone in the fluent.*

If f is a function symbol, then φ is independent of the fluents coding values of f unless f occurs in φ.

This lemma provides the basis for syntactic control of interference.

Domain constraints and non-frame fluents. Often, we are interested in actions operating on models for some first-order theory, rather than all abstract structures. For example, our theory might include the *domain constraint*, **switchModel** $\equiv$ isLight $\leftrightarrow$ isOn(switch1) $\oplus$ isOn(switch2), which allows either switch to toggle the light on/off. So, changes to the fluents isOn(switch1) or isOn(switch2) may entail changes to isLight, which must change value, if neccessary, to satisfy the constraint.

To represent this we can add **change** isLight to the changes of each ADL action, to allow changes to this *non-frame fluent*, and impose the domain constraint on the outcome by performing **require switchModel** before imposing the post-conditions of the action. In this way, the non-frame fluent may be used in the pre- and post-conditions of an action.

This approach can accommodate several non-frame fluents and numerous domain constraints, using one preparatory **change** and one conclusive **require** to bracket the changes of each action.

One challenge of knowledge representation for reasoning about action is to select appropriate frame fluents, which are generally subject to the law of inertia, and non-frame fluents, which are not—and whose values should therefore ordinarily be determined by those of the frame fluents.

A simple dichotomy between frame and non-frame fluents ignores the possibility that a given fluent may be changed, sometimes as a direct consequence of an action,

and sometimes as a consequence of other changes. A more refined approach would be to model causal relationships between fluents explicitly, as a directed acyclic graph, and allow a fluent to change (to accommodate domain constraints) whenever any of its ancestral causes changes. This would give a "poor logician's model" of causality (cf. [15]). A similar issue arises in the theory of database updates, where the analogue of domain constraints are called *integrity constraints*. For example, in [16], where Fagin *et. al.* suggest a prioritised hierarchy of integrity constraints, which treats the problem as an issue of belief revision, rather than update.

5 Appraisal

To return to the packing problem.

> *Before setting off on a trip, I pack a power adapter for my laptop (because I will need it and don't believe I have one packed already). Packing something ensures, non-deterministically, that it is in either my briefcase or my suitcase. I pack, place myself, briefcase and suitcase in the car, and drive the car from here to there. On arrival, I carry briefcase and suitcase from the car to my hotel room. Unpacking, I find not one, but two adapters, one in the briefcase, one in the suitcase. I realise that my earlier belief was mistaken: I* had *already packed an adapter. I hope my son won't mind that I have them both and he has none!*

We start with everything here: power-adapter, car, suitcase, briefcase, self. What features are required to represent and reason about this scenario? Packing the power supply is modelled by a non-deterministic combination of four classic STRIPS actions (either of the supplies in either of the cases). Putting the cases in the car is simple, a classic STRIPS action suffices for each.

When the car moves, things become more complicated. We can use a classic ADL conditional effect to account for the fact that the cases, and self, move with the car. The action

$$\begin{aligned}&\mathsf{Move(Z,X,Y)} = \mathsf{(require\ At(X,Z);}\\ &\qquad\qquad \mathsf{allow(not\ At(X,Z),\ At(Y,Z));require(not\ At(X,Z) \wedge At(Y,Z)))}\end{aligned} \quad (49)$$

requires a conditional effect

$$\begin{aligned}&\mathsf{(when\ In(Z,W)}\\ &\qquad \mathsf{(allow\ (not\ At(X,W),\ At(Y,W));\ require(not\ At(X,W) \wedge At(Y,W)))}\end{aligned} \quad (50)$$

To model the movement of the power adapters we use our extended ADL and add a further conditional effect

$$\begin{aligned}&\mathsf{(when\ (In(W,V) \wedge At(Y,W))}\\ &\qquad \mathsf{(allow\ (not\ At(X,V),\ At(Y,V));\ require(not\ At(X,V) \wedge At(Y,V)))}\end{aligned} \quad (51)$$

This does nothing without iteration—on the first iteration, the first of our conditional effects updates the location of the briefcase; on the second iteration the location of the power supply is changed. For this example, that is the end. The potential for arbitrarily deep nestings of objects means that we may need correspondingly many iterations—but we don't need to add any more conditional effects.

Progressing, through the sequence of actions leading to arrival at the hotel, an initial assumption that neither of the cases contains a power adapter, leads to a set of possible present states, none of which include two power adapters at the hotel. Regressing the observation of two adapters back down the sequence of actions, to the start of the story, leads to a set of states consistent with the current observation, all of which include an adapter packed in either one or other of the cases. Reconciling this knowledge with the earlier inconsistent belief is not a topic we will venture into here. However, just forgetting the earlier belief is one practical option. Progressing the new set back to the present leads to the useful conclusion that there are no adapters left at home.

This simple example requires all the features we have introduced. Our implementation [7] of the state logic uses BDDs to represent conditions. A standard BDD package can straightforwardly support all the operations described in the paper. These integrate smoothly with the use of BDDs to support planning (*op cit.*).

The *frame problem* [17] is a fundamental issue in the representation of actions. The STanford Research Institute Problem Solver (STRIPS) [5] was introduced as a computationally tractable solution to the frame problem. STRIPS has a clear and concise semantic basis, but is operational, rather than logical, in flavour. Moreover, only very simple actions can be formalised within the STRIPS framework.

Our treatment is formally close to that of Pednault [6]. In particular, we address the ramification problem through a combination of conditional effects, domain constraints, and non-frame fluents. However, our analysis allows us to extend its scope by introducing unbounded iteration to fixpoint for monotone actions.

The novel contributions here are: the separation of state-logic and domain-logic, which clarifies the state-transition semantics introduced by Pednault; the application of a unified logical treatment of update as bounded existential quantification to the static analysis of iteration and interference; the incorporation of non-determinism and concurrency; and the introduction of unbounded iteration as an extension of existing techniques for addressing the ramification problem.

6 Future Work

In this paper we use classical first-order predicate calculus as a domain logic—but more expressive logics with similar truth-functional semantics could be substituted for this straightforwardly, *mutatis mutandis*. It is also possible to extend this approach to modal domain logics, with Kripke-style semantics. An extension to a probabilistic account, in which a probability distribution on the set of states takes over the rôle played here by the subsets of state space, that we have called conditions, requires a new account of actions as functions from distributions to distributions [18]. Conditions play the role of *events*, to which a distribution assigns probabilities. The action **`require`** φ, where φ is a condition, has a direct analogue, corresponding to the operation of conditioning on

φ. The action **change** $\bar{x}$ is generalised to a family of operations, parametrised by the probability distributions on the finite set of possible values of $\bar{x}$.

References

1. Herzig, A., Lang, J., Marquis, P., Polacsek, T.: Updates, actions and planning. In Nebel, B., ed.: IJCAI'01, Seattle, Washington, USA, Morgan Kaufmann (2001) 119–124
2. Kartha, G.N., Lifschitz, V.: Actions with indirect effects (preliminary report). In Doyle, J., Sandewall, E., Torasso, P., eds.: KR'94: Principles of Knowledge Representation and Reasoning. Morgan Kaufmann, San Francisco, California (1994) 341–350
3. Sandewall, E.: Features and Fluents. Number 30 in Oxford Logic Guides. Oxford University Press (1994)
4. Lin, F., Reiter, R.: Forget it! In Greiner, R., Subramanian, D., eds.: Working Notes, AAAI Fall Symposium on Relevance, Menlo Park, California, American Association for Artificial Intelligence (1994) 154–159
5. Fikes, R., Nilsson, N.: STRIPS: a new approach to the application of theorem proving. Artificial Intelligence (1971) 189–208
6. Pednault, E.P.D.: ADL and the state-transition model of action. J. Logic Computat. **4**(5) (1994) 467–512
7. Fourman, M.P.: Propositional planning. In: Workshop on Model-Theoretic Aproaches to Planning, AIPS 2000. (2000) EDI-INF-RR-0034 `http://www.inf.ed.ac.uk/publications/report/0034.html`.
8. Pednault, E.P.D.: ADL: Exploring the middle ground between STRIPS and the situation calculus. In Ronald Brachman, Hector Levesque, R.R., ed.: Proc. First Int'l Conf. on Principles of Knowledge Representation and Reasoning. (1989) 324–332
9. Shanahan, M.: The frame problem. In Zalta, E.N., ed.: The Stanford Encyclopedia of Philosophy. Stanford University (Spring 2006) URL = `http://plato.stanford.edu/archives/spr2006/entries/frame-problem/`.
10. Marx, K.: Theses on Feuerbach. In: Ludwig Feuerbach and the End of Classical German Philosophy, 1886;. Volume 1 of Marx/Engels Selected Works. Progress Publishers (1845)
11. Lang, J., Liberatore, P., Marquis, P.: Propositional independence: Formula-variable independence and forgetting. JAIR **18** (2003) 391–443
12. Lang, J., Marquis, P.: Complexity results for independence and definability in propositional logic. In Cohn, A.G., Schubert, L., Shapiro, S.C., eds.: KR'98: Principles of Knowledge Representation and Reasoning, San Francisco, California, Morgan Kaufmann (1998) 356–367
13. Edelkamp, S., Helmert, M.: MIPS: The model-checking integrated planning system. AI Magazine **22**(3) (2001) 67–72
14. Mueller, E.T.: Event calculus reasoning through satisfiability. J Logic Computation **14**(5) (2004) 731–745
15. Pearl, J.: Causality: Models, Reasoning, and Inference. Cambridge University Press (2000)
16. Fagin, R., Ullman, J.D., Vardi, M.Y.: On the semantics of updates in databases. In: Proc. Second ACM SIGACT-SIGMOD, ACM (1983) 352–365
17. McCarthy, J., Hayes, P.J.: Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., Michie, D., eds.: Machine Intelligence 4. Edinburgh University Press (1969) 463–502 reprinted in [19].
18. Fourman, M.P.: Logics for action and belief. in preparation (2007)
19. McCarthy, J.: Formalization of common sense, papers by John McCarthy edited by V. Lifschitz. Ablex (1990)