Executability in the Situation Calculus

Timothy Cerexhe and Maurice Pagnucco

ARC Centre of Excellence in Autonomous Systems,
School of Computer Science and Engineering, UNSW,
Sydney, NSW, 2052, Australia
{timothyc,morri}@cse.unsw.edu.au

Abstract. This paper establishes a formal relationship between theories of computation and certain types of reasoning about action theories expressed in the situation calculus. In particular it establishes a formal correspondence between Deterministic Finite-State Automata (DFAs) and the 'literal-based' class of basic action theory, and identifies the special case of DFAs equivalent to 'context-free' action theories. These results formally describe the relative expressivity of different action theories. We intend to exploit these results to drive more efficient implementations for planning, legality checking, and modelling in the situation calculus.

Keywords: Reasoning about Action and Change, Situation Calculus, Theory of Computation.

1 Introduction

The *situation calculus* is a general-purpose dialect of first-order logic for reasoning about the effects of actions in dynamic domains. It allows us to model the state of a changing world, or determine which sequence of actions will achieve a goal. However not all sequences will be executable—the effects of one action may violate the preconditions of another. In this light, the set of executable sequences of actions for a given theory corresponds to the more general concept of a language whose strings are built from an alphabet of actions.

Executability in the situation calculus has been briefly defined and discussed by Reiter [5]. However there has been no attempt to describe the complexity of an action theory based on the language of sequences of actions it accepts. This is surprising since, compared with logic-based action calculi, automata theory provides a powerful and more thoroughly studied means of classifying the 'hardness' of different computational machinery—including those required for recognising or constructing action theories.

In this paper we identify several existing special cases of action theories in the situation calculus and prove their equivalence with classes of deterministic finite automata. This facilitates a greater understanding of the expressivity of situation calculus theories. We also expect it to lead to more efficient techniques for representing and manipulating action theories, such as in planning problems or in new interpreters for the cognitive robotics language Golog [1] whose semantics are based on the situation calculus. By the same token, our translations allow AI behaviours encoded via finite state machines—such as XABSL [3] used for robotics—to be converted to the situation calculus where it may be formally reasoned about.

2 Background

The situation calculus is a sorted first-order logic for reasoning about actions in dynamic systems with several distinguished elements. *Situation* terms are histories (sequences of actions) composed of the binary $do(a, s)$ function which returns the situation that results from performing action a in situation s. Thus $do(a_n, \ldots, do(a_2, do(a_1, S_0)))$ represents the situation after performing the actions $a_1, a_2, \ldots, a_n$ in that order, starting from the distinguished *initial* situation S_0. Arbitrary situations can be constructed this way. *Fluents* are functions that represent properties of the world. Their value may be modified by performing actions, so they have situation terms as their last argument. We require that the initial situation be fully axiomatised—every fluent must have a known value in S_0. *Precondition axioms* defining $poss(a, s)$ specify the conditions under which action a may be performed in situation s. *Effect axioms* specify the resulting fluent values after performing an action a in situation s.

A situation-suppressed fluent f represents the partial function $f(s)$ without its situation term. A situation-independent formula does not expect a situation term. A formula is *uniform in s* if it does not mention *poss* or *do*, or quantify over situations, and s is the only situation term that occurs. Essentially it restricts the use of the situation terms to querying the current value of fluents in only that situation.

Note that rather than use effect axioms we adopt Reiter's *successor state axioms* (SSAs)—one per fluent—that provide a solution to the frame problem.[1] SSAs can be automatically generated from effect axioms. This gives the following 'template' for successor state axioms:

$$f(do(a, s)) = y \equiv \gamma^+(a, y, s) \lor (f(s) = y \land \neg\exists z \,.\, \gamma^-(a, z, s))$$

where γ^+, γ^- are the positive and negative effects respectively—the conditions that describe when a value becomes true or false. This formula expresses that fluent f has the value y after performing action a in situation s whenever the conditions that make the fluent assume that value hold (γ^+) or f already has the value y and nothing occurs to make it false (γ^-). Note that we will only consider functional fluents in this paper, so a positive effect for one value must coincide with negative effects for all other values in the domain of that fluent.

The above SSA describes the value of fluent f in situation $do(a, s)$ given its previous value $f(s)$ and the conditions that update its value (γ^+) or the absence of effects that make it false (γ^-). This last component provides 'inertia'—the

[1] A discussion of the frame problem is beyond the scope of this paper, suffice to say that it represents an explosion of axioms required to logically model fluents in dynamic systems that do not change their value as a result of performing an action.

logical assertion that fluents are not affected by unrelated actions. For example the colour of a block does not change as a result of picking it up.

Systems always start in the distinguished initial situation S_0. The value of a fluent f in a later situation s can be determined by regressing[2] back to S_0 and simulating the update axioms on f for each action in s. We are now in a position to define a *basic action theory* which represents a rudimentary type of action theory.

Definition 1. *A finite basic action theory (BAT) is a bounded theory in the situation calculus $\mathcal{B} = \Sigma_{sitcalc} \cup \mathcal{B}_{una} \cup \mathcal{B}_{ss} \cup \mathcal{B}_{ap} \cup \mathcal{B}_{S_0}$, where:*

1. *$\Sigma_{sitcalc}$ are the foundational axioms for the situation calculus that logically describe what a situation looks like.*
2. *$\mathcal{B}_{una}$ are unique names assumptions—logical assertions that distinct names (including fluent, action, and object names) are treated as distinct concepts.*
3. *$\mathcal{B}_{ss}$ is a finite set of successor state axioms of the form above.*
4. *$\mathcal{B}_{ap}$ is a finite set of action precondition axioms of the form $poss(a, s) \equiv \Pi_a(s)$, one for each action. Π_a must be uniform in s.*
5. *$\mathcal{B}_{S_0}$ is a finite set of first-order sentences defining the initial situation S_0.*
6. *There are three finite sets of fluent names, actions, and objects. All ground fluents and actions are constructed from these sets. We shall refer to the sets of* ground *terms as $\mathcal{F}$, $\mathcal{A}$, and $\mathcal{O}$, and assume that each has an arbitrary but fixed ordering.*

For an arbitrary machine M, we say its language $\mathcal{L}(M)$ is some (possibly infinite) set of accepted words $w \in \mathcal{L}(M)$. We will consider a BAT of the situation calculus to be one such machine, as well as more conventional machines like Deterministic Finite-State Automata (DFAs). Both machines depend on a transition mechanism. DFAs use a function δ to map states and symbols to new states. The situation calculus analogue is the function *do* that maps situations and actions to new situations.

Definition 2. *A DFA D is a 5-tuple:*

$$D = \langle Q, \Sigma, q_0, \delta, F \rangle$$

where Q is a finite, non-empty set of states, Σ is the input alphabet. The (total) transition function $\delta : Q \times \Sigma \rightarrow Q$ maps a state and a symbol to a new state. The system starts in state q_0 and transitions according to δ. The set $F \subseteq Q$ identifies the final (accepting) states of Q. If the system reaches one of these states it may *accept—the current string of symbols is a word in the language—or continue running to accept a longer string.*

The transition function of an automaton encapsulates the pre- and post- conditions of a transition. These are separated in the situation calculus as action

[2] See Reiter [5]. We leave the alternative method—progression—for future work.

preconditions (*poss*) and fluent successor state axiom postconditions. It is relevant then to define *executable* situations as a sequence of actions where one action's postconditions are sufficient to guarantee the preconditions of the next:

$$exec(do(a,s)) \equiv exec(s) \land poss(a,s) \tag{exec}$$

and

$$\mathcal{B} \cup (exec) \models exec(s)$$

This is one of Reiter's definitions of executability. Note that $exec(S_0)$ is considered trivially true, though it is also easily derivable from alternate (equivalent) definitions [5].

Many automata explicitly define a set of final states. In contrast, the situation calculus makes no such distinction. To this end, we introduce an artificial finality condition—a situation s is *final* iff a distinguished formula, $f_{halt}(s)$, is true in that situation. This has applications to planning problems, where the goal is often specified by a formula $goal(s)$ identifying final states/situations. It is necessary in this paper for f_{halt} to be uniform in s. This would restrict goal formulas to being Markovian.

Finally, we define acceptance as:

Definition 3. *A DFA accepts a sequence if the corresponding transitions produce a final state. A BAT accepts a situation (sequence of actions) if that situation is executable and satisfies some theory-specific halt (goal) condition.*

$$\begin{aligned} w \in \mathcal{L}(DFA) &\text{ iff } \hat{\delta}(q_0, w) \in F \\ w \in \mathcal{E}(BAT) &\text{ iff } BAT \models f_{halt}(\hat{do}(w, S_0)) \land exec(\hat{do}(w, S_0)) \end{aligned}$$

where $\hat{\delta}, \hat{do}$ represent repeated applications of the corresponding transition functions.

Informally, $\mathcal{L}(D)$ is the set of words that transition to a final state in DFA D. $\mathcal{E}(B)$ is the set of legal, 'halting' situations in BAT B—to distinguish it from logically derivable sentences $\mathcal{L}$ in a typical knowledge base.

3 DFA—Literal-Based BAT Equivalence

In this section we prove the formal equivalence between DFAs and the 'literal-based' class of basic action theories. Petrick and Levesque defined literal-based BATs while establishing knowledge equivalence [4]. The functional form was left as an exercise and so we provide the following definition:

Definition 4. *A literal-based BAT (LB-BAT) is a finite BAT of the above form, but with restricted use of the situation term in the positive and negative effects (γ^+, γ^-) of the successor state axioms:*

$$\gamma_F^+(y,a,s),\ \gamma_F^-(y,a,s) \overset{def}{=} \bigvee_{i=1}^{k} \pi_i(y,a,s)$$

$$\textit{where each } \pi_i \textit{ is:} \qquad \pi_i(y,a,s) \overset{def}{=} \exists \boldsymbol{z}_i \,.\, a = \beta_i(\boldsymbol{z}_i) \wedge \psi_i(y,\boldsymbol{z}_i,a) \wedge P_1(\boldsymbol{z}_i,s) = c_1 \wedge \ldots \wedge P_l(\boldsymbol{z}_i,s) = c_l$$

where $P_1,\ldots,P_l$ are fluent literals, $c_1,\ldots,c_l$ are constants, the (possibly empty) vector of variables $\boldsymbol{z}_i$ must be an argument to the action term β_i. To maintain the functional property, we also require that all separate πs are mutually inconsistent.

Informally, SSAs may now only use the situation term to mention a finite conjunction of fluent literals, or to implement inertia.

Fluent values are typically encapsulated in situations, which may grow arbitrarily long. However literal-based BATs have finite domains, so we can define equivalence classes of situations that represent the finite state space of distinct fluent values. This formulation is suitable for an automaton.

Definition 5. *state is a function from* executable *situations in BAT B to ordered tuples of fluent values:*

$$state(s) = \begin{cases} \langle\, \mathcal{B} \models f(s)\,\rangle_{f\in\mathcal{F}} & \textit{if } \mathcal{B} \models exec(s) \\ \bot & \textit{otherwise} \end{cases}$$

where $\mathcal{F}$ is the finite (ordered) set of unique ground fluents in B. Note that non-executable *situations all map to a distinguished failure state $\bot$.*

Definition 6. *The* BAT-state construction *takes a literal-based basic action theory B and produces a finite state machine $D = \langle Q, q_0, \Sigma, \delta, F_D\rangle$ where:*

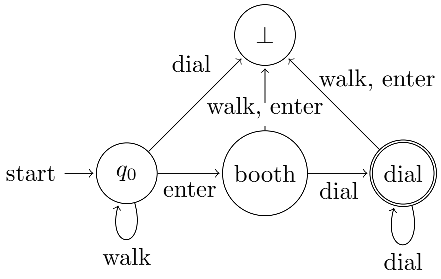
$$\begin{aligned} Q &= \{state(s) \,|\, s \textit{ is an executable situation}\} \cup \{\bot\} && \text{(set of states)} \\ q_0 &= state(S_0) && \text{(initial state)} \\ \Sigma &= \textit{the set of ground action terms} && \text{(alphabet)} \\ \delta(state(s),a) &= state(do(a,s)) && \text{(transition function } Q\times\Sigma\rightarrow Q\text{)} \\ \delta(\bot,a) &= \bot && \\ F_D &= \{state(s)\,|\,\mathcal{B} \models f_{halt}(s)\} && \text{(final states } \subseteq Q\text{)} \end{aligned}$$

Note also that *state* maps infinitely-many situations (histories) to finitely-many 'states'; the domain of each fluent $dom(f)$ is finite, and both literal-based preconditions and successor-state axioms are finite and uniform. The set of states Q in our DFA is an equivalence class of situations with the same fluent-value bindings.

We illustrate this conversion with an example. Our hero, Maxwell, walks down a corridor full of doors on his way to work. At the end of the corridor is a phone booth in which he dials a passcode. Such a procedure may be represented as a literal-based action theory where we keep walking until we non-deterministically decide that we have reached the phone, at which point we enter and dial. This would typically be augmented with sensing actions, though we prefer to save a discussion of this mechanism for Section 5.

actions:	$\{walk,\ enter,\ dial\}$
fluents:	$\{dialled,\ in\_booth\}$
initially:	$dialled(S_0) = F$ $in\_booth(S_0) = F$
termination:	$f_{halt}(s) \equiv dialled(s)$
preconditions:	$poss(walk, s) \equiv \neg in\_booth(s)$ $poss(enter, s) \equiv \neg in\_booth(s)$ $poss(dial, s) \equiv in\_booth(s)$
SSAs:	$dialled(do(a, s)) = y \equiv (a = dial \wedge y = T) \vee (dialled(s) = y \wedge a \neq dial)$ $in\_booth(do(a, s)) = y \equiv (a = enter \wedge y = T) \vee (in\_booth(s) = y \wedge a \neq enter)$

(a) An LB-BAT for 'corridor world'



(b) Corresponding DFA

Fig. 1. Corridor World

We can now apply our construction to this action theory. First, our alphabet is the set of actions $\Sigma = \{walk, enter, dial\}$. Each fluent has a Boolean domain, so our state-space is:

$$Q = \{\langle\rangle, \langle in\_booth\rangle, \langle dialled, in\_booth\rangle, \perp\}$$

For clarity we represent a state as the tuple of true fluents in that state. The initial situation has both fluents false, that is $q_0 = \langle\rangle$. Termination is defined as any states that have *dialled* true:

$$F_D = \{\langle dialled, in\_booth\rangle\}$$

Finally, the update function maps the following states:

$$\delta(\langle\rangle, walk) = \langle\rangle \qquad \delta(\langle in\_booth\rangle, dial) = \langle dialled, in\_booth\rangle$$
$$\delta(\langle\rangle, enter) = \langle in\_booth\rangle \qquad \delta(\langle dialled, in\_booth\rangle, dial) = \langle dialled, in\_booth\rangle$$

All other transitions are illegal (violate *poss* axioms).

Theorem 1. *If B is a literal-based BAT and D is the DFA obtained by applying the BAT-state construction on B, then $\mathcal{E}(B) = \mathcal{L}(D)$.*

We propose a similar construction in the reverse direction:

Definition 7. *The* fluent construction *takes a DFA D and produces a literal-based BAT $B = \Sigma_{sitcalc} \cup \mathcal{B}_{ss} \cup \mathcal{B}_{ap} \cup \mathcal{B}_{una} \cup \mathcal{B}_{S_0}$ where:*

1. *$flu : Q \rightarrow \mathcal{F}$ maps states to unique fluent names.*
2. *$\Sigma_{sitcalc} \equiv$ situation calculus foundational axioms.*

3. $\mathcal{B}_{ss} \equiv \bigcup_{f \in flu} SSA_f$*; the effect axioms are the set of SSAs for each fluent name in* flu*.*
4. $\mathcal{B}_{ap} \equiv \bigcup_{a \in \Sigma_D} poss(a,s)$*; the action preconditions are the set of poss axioms for each action in the DFA alphabet* Σ_D*.*
5. $poss(a,s) \equiv \bigvee_q flu_q(s) = T$ *where* $\delta(q,a) \neq \bot$*; action* a *is possible in any state where it won't transition to the sink.*
6. $\gamma_f^+(T,\alpha,s) \equiv \gamma_f^-(F,\alpha,s) \equiv \bigvee_q (\alpha = a \wedge flu_q(s) = T)$ *where* $f = flu_{\delta(q,a)}$*; action* a *makes fluent* f *true if the current* f' *state* ($q \Rightarrow f'(s)$) *would transition to the* f *state* ($\delta(q,a) \Rightarrow f(do(a,s))$)*.*
7. $\gamma_f^-(F,\alpha,s) \equiv \gamma_f^+(T,\alpha,s) \equiv$ *dual of* $\gamma_f^+(F,\alpha,s)$*.*
8. $\mathcal{B}_{S_0} \equiv$ *initial situation. The initial state's fluent is true:* $flu_{q_0}(S_0) = T$*. All other fluents are false:* $f(S_0) = F$*. Also, the initial situation is final iff the initial state is:* $f_{halt}(S_0) \equiv q_0 \in F_D$*.*
9. $f_{halt}(s) \equiv \bigvee_{q \in F_D} flu_q(s) = T$*; the halt condition holds iff any final state's fluent does.*

And $\mathcal{B}_{una}$ *ensures that the actions* $a \in \Sigma_D$ *for distinct transition labels and the fluents* $f \in flu \cup \{f_{halt}\}$ *for distinct states are all logically unique.*

Note that the preconditions are uniform, the successor state axioms are uniform and literal-based, and $f_{halt} \notin flu$.

Theorem 2. *If* D *is a DFA and* B *is the literal-based BAT obtained by applying the fluent construction on* D*, then* $\mathcal{L}(D) = \mathcal{E}(B)$*.*

Corollary 1. *DFAs and LB-BATs are equivalent:*

$$\mathcal{L}(DFA) = \mathcal{E}(LB\text{-}BAT).$$

4 Lattice DFA–Context-Free BAT Equivalence

The 'context free' successor state axioms are a weak special case of the literal-based SSAs. This means that the DFA construction in Definition 6 still applies. The DFA that we get will be severely restricted though—a visual indication of the restrictiveness of context-free BATS (CF-BATs). We identify this class as 'Lattice' DFAs because of their high geometric symmetry, and prove their equivalence to CF-BATs.

Definition 8. *A context-free BAT (CF-BAT) is a special case of LB-BAT with additional restrictions on SSAs—the positive and negative effects* (γ^+, γ^-) *can have no situation dependence:*

$$f(do(a,s)) = y \equiv \gamma_f^+(a,y) \vee (f(s) = y \wedge \neg\exists z \,.\, \gamma_f^-(a,z))$$

Definition 9. *A Lattice DFA is a DFA whose states can be partitioned in one or more ways such that all reachable transitions satisfy the following conditions:*

1. *The transitions in a partition wrt an action (an $(action, partition)$ pair) must be 'fixed' or 'inertial':*
   (a) Fixed — all states transition into the same part.
   (b) Inertial — no state can transition between parts.
2. *If every partition is Inertial wrt action a, then a can only label self-loops.*
3. *If every partition is Fixed wrt action a, then a must always transition to the same state.*
4. *There cannot be more partitions than states.*
5. *There cannot be more parts in a partition than the number of actions.*
6. *No two states can appear in the same part across every partition (indistinguishable).*

All other transitions must enter the inescapable sink state $\perp$. Note that Conditions 2 and 3 motivate multiple partitions for most useful Lattice DFAs.

Each partition is a distinct view of the same set of states and represents how states are distinguishable under action transitions—both as a dimension of symmetry in the DFA, and the domain of a fluent in the corresponding CF-BAT.

The following heuristic is useful for categorising $(action, partition)$ pairs:

1. If action a transitions between two distinct parts of partition p, then p must be Fixed wrt a;
2. Otherwise, if there are transitions starting within two separate parts of partition p, then p must be Inertial wrt a; and,
3. Otherwise, it can be either, subject to Conditions 2+3.

Definition 10. *$part_f$ takes a state and returns the index of its part along the fth partition:*

$$part_f : Q \to \mathbb{N}$$

Note that the conjunction of part indices across all partitions uniquely defines the state, $q \equiv \bigwedge_f part_f(q)$.

The Lattice DFA is heavily restricted and impractical as a modelling tool, but we show that it can represent any context-free BAT. Note first that the BAT-state construction from Definition 6 is sufficient to convert CF-BATs into DFAs since CF-BATS are a special case of LB-BAT. The partitions of the DFA in this case correspond directly to the fluents. If you construct a DFA from a CF-BAT with n fluents, such that states with the same fluent-value all lie on an $(n-1)$-dimensional hyperplane, then you get an n-dimensional lattice. The distinct parts of each partition (the hyperplanes) represent the domain of the corresponding fluent. This geometrical interpretation represents the restrictions that context-free SSAs place on transitions—we *can not* compile out the fluents, so instead the transitions exhibit high dimensional symmetry, and the fluents remain partially represented in the physical construction of the automata.

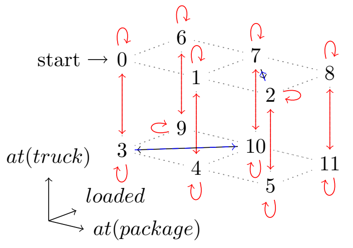


Fig. 2. A 3D Lattice DFA

Table 1. The (*action, partition*) types

		partition		
		loaded	*at(package)*	*at(truck)*
action	$drop_{depot}$	fixed	fixed	either
	pickup	fixed	fixed	inertial
	$drop_{Perth}$	fixed	fixed	either
	$drive_{depot}$	inertial	inertial	fixed
	$drive_{Perth}$	inertial	inertial	fixed

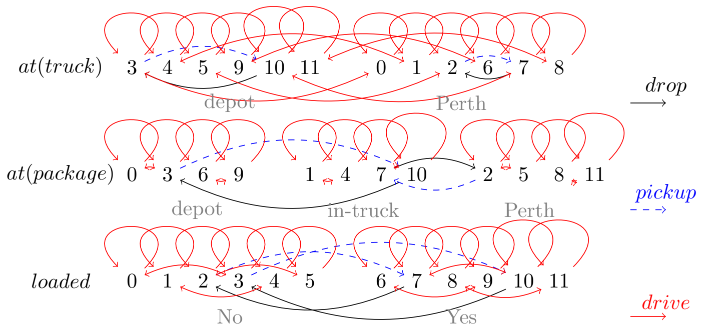


Fig. 3. The partitions of the DFA

We illustrate with a reduced Depot Problem. The part labels in Figure 3 (eg. *Perth* under *at*(*truck*)) indicate that the start state (0) has the truck at Perth and the package unloaded, at the depot. Note that all *drive*-transitions stay in their original part in the *loaded* and *at*(*package*) partitions and hence are inertial—the *drive* action does not affect these fluents. Conversely, the two *drop* actions may be either inertial or fixed on the *at*(*truck*) partition as the truck's location does not change (inertial), but may equivalently be *set* by the action effects (fixed).

Theorem 3. *If B is a context-free BAT and D is the DFA obtained by applying BAT-state construction on B, then D will be a Lattice DFA and $\mathcal{L}(D) = \mathcal{E}(B)$.*

Definition 11. *The Lattice-BAT construction takes a Lattice DFA D and produces a context-free BAT B such that:*

1. *$actions = \Sigma_B$*
2. *$fluents = \{f_i \,|\, 0 \leq i < |partitions\ of\ D|\}$*
3. *$poss(a, s) \equiv \bigvee_{q \in Q} \bigwedge_i f_i(s) = part_{f_i}(q) \wedge \delta(q, a) \neq \bot$*
4. *$f_i(do(a, s)) = y \equiv \gamma^+_{f_i}(a, y) \vee (f_i(s) = y \wedge \neg\exists z \,.\, \gamma^-_{f_i}(a, z))$*
   where $\gamma^+_{f_i}(a, y) \equiv \exists q \,.\, \delta(q, a) \neq q \wedge \delta(q, a) \neq \bot \wedge part_{f_i}(\delta(q, a)) = y$
   $\gamma^-_{f_i}(a, y) \equiv$ dual of $\gamma^+_{f_i}$ if it exists

5. $\mathcal{D}_{S_0} \equiv \{f_i(S_0) = part_{f_i}(q_0) \,|\, 0 \leq i < |partitions\ of\ D|\}$
6. $f_{halt}(s) \equiv \bigvee_{q \in F_D} \bigwedge_{f_i} f_i(S_0) = part_{f_i}(q)$.

Note that the γ^+ construction only exists if the transition has a change of state — a Fixed transition. Inertial transitions are handled by the absence of γ^+ and γ^- axioms for that action. There can be at most one γ^+ and one γ^- for an action because the arguments do not provide greater granularity. This is why each $(action, partition)$ pair must satisfy either the Fixed or Inertial requirements.

Theorem 4. *Every Lattice DFA D can be converted to an equivalent context-free BAT B such that $\mathcal{L}(D) = \mathcal{E}(B)$.*

Corollary 2. *Lattice DFAs and CF-BATs are equivalent:*

$$\mathcal{L}(Lattice\ DFA) = \mathcal{E}(CF\text{-}BAT).$$

5 Conclusion

The situation calculus contains other types of actions beside the 'primitive' actions used above. Sensing actions update fluents directly based on a sensed value, rather than by an effect axiom. Exogenous actions are actions performed externally, but whose effects must be detected so that the internal model can remain consistent with the state of the actual world.

These are technically extra-automata features, however, the situation calculus model is from 'god's eye' or meta view—we assume the logical state mirrors the world it models. In this light exogenous actions are simply regular actions fired by a different hand—a distinction that is irrelevant from an automaton perspective. Similarly, sensing actions can be modeled by non-deterministically selecting a sensing action that returns the correct result. Introducing Lin's indeterminate effects axioms [2] to the situation calculus should facilitate this aspect.

We also intend to investigate the application of these results to planning problems. We believe that analysing special cases like the Lattice DFA will help identify tractable domains.

References

1. Levesque, H.J., Reiter, R., Lespérance, Y., Lin, F., Scherl, R.B.: GOLOG: A logic programming language for dynamic domains. J. Log. Program. 31(1-3), 59–83 (1997)
2. Lin, F.: Embracing causality in specifying the indeterminate effects of actions. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence, AAAI 1996, vol. 1, pp. 670–676. AAAI Press (1996)
3. Lötzsch, M., Bach, J., Burkhard, H.-D., Jüngel, M.: Designing Agent Behavior with the Extensible Agent Behavior Specification Language XABSL. In: Polani, D., Browning, B., Bonarini, A., Yoshida, K. (eds.) RoboCup 2003. LNCS (LNAI), vol. 3020, pp. 114–124. Springer, Heidelberg (2004)
4. Petrick, R.P.A., Levesque, H.J.: Knowledge equivalence in combined action theories. In: KR 2002, pp. 303–314 (2002)
5. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. illustrated edn. The MIT Press (September 2001)