

First-Order Strong Progression for Local-Effect Basic Action Theories

Stavros Vassos¹, Gerhard Lakemeyer², Hector J. Levesque¹

¹Department of Computer Science
University of Toronto



²Department of Computer Science
RWTH Aachen University



KR 2008



Introduction

The problem we examine lies in the field of *reasoning about action and change*.

Given a logical formalism that is able to:

- 1 represent the current state of the world;
- 2 represent the dynamics of the world;
- 3 answer queries about the current state and the possible future states of the world.

Problem: How to update the representation of the current state after action execution.



Introduction



Think of a non-player character (NPC) in video game equipped with such a logical formalism.

1 Representing the current state:

- ▶ “the door is locked”
- ▶ “the red and green levers are up and the yellow lever is down”



2 Representing the dynamics of the world:

- ▶ “pushing/pulling a lever toggles its up/down position”
- ▶ “pressing the button unlocks the door, provided that all levers are in up position”

3 Answering queries about the future:

- ▶ “is there a sequence of actions such that after executing it the door will be unlocked?”



Basic action theories in the situation calculus

The *situation calculus* is a first-order logical language with limited second-order features:

- S_0 is the initial situation;
- $do(a, S_0)$ is the resulting situation after action a has been performed in S_0 ;
- *fluents* are like normal predicates but also depend on a situation argument:
 $locked(S_0)$,
 $\neg leverUp(yellow, do(a, S_0))$.



Basic action theories in the situation calculus

The *situation calculus* is a first-order logical language with limited second-order features:

- S_0 is the initial situation;
- $do(a, S_0)$ is the resulting situation after action a has been performed in S_0 ;
- *fluents* are like normal predicates but also depend on a situation argument:
 $locked(S_0)$,
 $\neg leverUp(yellow, do(a, S_0))$.

A *basic action theory (BAT)* consists essentially of:

- **KB**: a first-order knowledge base (possibly with quantifiers) that represents what holds in the initial situation S_0 ;
- **DYN**: a set of action precondition and effect axioms that represent the dynamics of the world.



Problem: progression of basic action theories

Queries about the future are formed as entailment questions:

- $BAT \models \text{locked}(\text{do}(a_1, S_0))$ $[a_1]$
- $BAT \models \neg \text{locked}(\text{do}(a_2, \text{do}(a_1, S_0)))$ $[a_1, a_2]$



Problem: progression of basic action theories

Queries about the future are formed as entailment questions:

- $BAT \models \text{locked}(\text{do}(a_1, S_0))$ $[a_1]$
- $BAT \models \neg \text{locked}(\text{do}(a_2, \text{do}(a_1, S_0)))$ $[a_1, a_2]$

The NPC chooses to *execute* actions $[a_1, a_2]$ that result in unlocking the door.

We want a *new* BAT that *progresses* S_0 to the current state.

This is necessary for *long-lived* agents, like our NPC.



Problem: progression of basic action theories

BAT : KB (knowledge base about S_0) + DYN (axioms for dynamics),
action a is executed by the robot.

BAT': *new* KB' + *same* DYN,
such that *BAT'* *entails* the same *first-order* sentences about
the *future* (after a has been executed) as *BAT*.



Problem: progression of basic action theories

BAT : KB (knowledge base about S_0) + DYN (axioms for dynamics),
action a is executed by the robot.

BAT': *new* KB' + *same* DYN,
such that *BAT'* *entails* the same *first-order* sentences about
the *future* (after a has been executed) as *BAT*.

BAT	BAT' (progression)
unrestricted KB unrestricted DYN	KB' needs to be second-order! [Lin & Reiter 97], [Vassos & Levesque 08]
unrestricted KB <i>restricted</i> DYN	KB' first-order representable assuming a practical restriction, <i>local-effects</i> ?



Local-effect BATs

A BAT is *local-effect* iff actions may only affect the truth value of fluent atoms where arguments are also arguments of the action.

- Action *mess-with*(*c*, *d*) may only affect ground fluent atoms with arguments in {*c*, *d*}.
- Action *push*(*x*) that is intended to affect the truth value of *leverUp*(*x*, *s*) is local-effect.
E.g., *push*(*green*) may only affect *leverUp*(*green*, *s*).
- Action *push-all* that is intended to affect all three levers is not local-effect.

Note that this is a restriction on DYN only.



Result 1 (local-effect): Progression is first order

BAT : *unrestricted* KB + *local-effect* DYN ,
action a is executed by the agent.

BAT' : *first-order* KB' + *same* DYN.

Theorem: For any local-effect BAT and any ground action a there is always a first-order representation of the new (progressed) KB'.



Result 1 (local-effect): Progression is first order

BAT : *unrestricted* KB + *local-effect* DYN ,
action a is executed by the agent.

BAT' : *first-order* KB' + *same* DYN.

Theorem: For any local-effect BAT and any ground action a there is always a first-order representation of the new (progressed) KB'.

KB' is equivalent to the (*infinite*) set of first-order entailments of BAT that only talk about situation $do(a, S_0)$;



Result 1 (local-effect): Progression is first order

BAT : *unrestricted* KB + *local-effect* DYN ,
action a is executed by the agent.

BAT' : *first-order* KB' + *same* DYN.

Theorem: For any local-effect BAT and any ground action a there is always a first-order representation of the new (progressed) KB'.

KB' is equivalent to the (*infinite*) set of first-order entailments of BAT that only talk about situation $do(a, S_0)$;

If we restrict the BATs a little more then we can specify a *finite* first-order KB'.



Strictly Local-effect BATs

A BAT is *strictly local-effect* iff it is local-effect and moreover:

- KB is finite and has unique-names for objects;
- DYN is further restricted so that conditional effects cannot quantify over fluent atoms in the condition.

Example: action *pushbutton* will result in unlocking the door provided all levers are up.

Two ways to formalize the condition “all levers are up”:

- Local-effect: *pushbutton* affects *locked(s)* provided

$$\forall x(\text{lever}(x) \supset \text{leverUp}(x, s))$$

- Strictly local-effect: *pushbutton* affects *locked(s)* provided

$$\text{leverUp}(\text{red}, s) \wedge \text{leverUp}(\text{green}, s) \wedge \text{leverUp}(\text{yellow}, s)$$



Result 2 (strictly local-effect): Progression is finite

BAT : *finite, unique names* KB + *strictly local-effect* DYN,
action a is executed by the agent.

BAT': *finite, first-order* KB' + *same* DYN.

Theorem: For any strictly local-effect BAT and any ground action a we can always specify a finite first-order representation of KB'.



Result 2 (strictly local-effect): Progression is finite

BAT : *finite, unique names* KB + *strictly local-effect* DYN,
action a is executed by the agent.

BAT': *finite, first-order* KB' + *same* DYN.

Theorem: For any strictly local-effect BAT and any ground action a we can always specify a finite first-order representation of KB'.

KB' can be obtained by modifying the original KB in a systematic way.



Result 2 (strictly local-effect): Progression is finite

BAT : *finite, unique names* KB + *strictly local-effect* DYN,
action *a* is executed by the agent.

BAT': *finite, first-order* KB' + *same* DYN.

Theorem: For any strictly local-effect BAT and any ground action *a* we can always specify a finite first-order representation of KB'.

KB' can be obtained by modifying the original KB in a systematic way.

Let's see an example!



Result 2 (strictly local-effect): Example

- KB: $\forall x F(x, S_0)$
- DYN: action $a(x)$ flips the value of $F(x, s)$

Execute action $a(c)$ and progress by modifying the original KB.



Result 2 (strictly local-effect): Example

- KB: $\forall x F(x, S_0)$
- DYN: action $a(x)$ flips the value of $F(x, s)$

Execute action $a(c)$ and progress by modifying the original KB.

1 $a(c)$ may only affect the value of $F(c, S_0)$. Consider cases:

$$F(c, S_0) \wedge KB \quad \vee \quad \neg F(c, S_0) \wedge KB$$



Result 2 (strictly local-effect): Example

- KB: $\forall x F(x, S_0)$
- DYN: action $a(x)$ flips the value of $F(x, s)$

Execute action $a(c)$ and progress by modifying the original KB.

1 $a(c)$ may only affect the value of $F(c, S_0)$. Consider cases:

$$F(c, S_0) \wedge KB \quad \vee \quad \neg F(c, S_0) \wedge KB$$

2 For each case unit-propagate “in a first-order way”:

$$F(c, S_0) \wedge \forall x (x = c \wedge F(x, S_0) \vee x \neq c \wedge F(x, S_0)) \quad \vee \\ \neg F(c, S_0) \wedge \forall x (x = c \wedge F(x, S_0) \vee x \neq c \wedge F(x, S_0))$$



Result 2 (strictly local-effect): Example

- KB: $\forall x F(x, S_0)$
- DYN: action $a(x)$ flips the value of $F(x, s)$

Execute action $a(c)$ and progress by modifying the original KB.

1 $a(c)$ may only affect the value of $F(c, S_0)$. Consider cases:

$$F(c, S_0) \wedge KB \quad \vee \quad \neg F(c, S_0) \wedge KB$$

2 For each case unit-propagate “in a first-order way”:

$$F(c, S_0) \wedge \forall x (x = c \wedge \text{true} \quad \vee \quad x \neq c \wedge F(x, S_0)) \quad \vee$$
$$\neg F(c, S_0) \wedge \forall x (x = c \wedge \text{false} \quad \vee \quad x \neq c \wedge F(x, S_0))$$



Result 2 (strictly local-effect): Example

- KB: $\forall x F(x, S_0)$
- DYN: action $a(x)$ flips the value of $F(x, s)$

Execute action $a(c)$ and progress by modifying the original KB.

1 $a(c)$ may only affect the value of $F(c, S_0)$. Consider cases:

$$F(c, S_0) \wedge KB \quad \vee \quad \neg F(c, S_0) \wedge KB$$

2 For each case unit-propagate “in a first-order way”:

$$F(c, S_0) \wedge \forall x (x = c \wedge \text{true} \vee x \neq c \wedge F(x, S_0)) \quad \vee \\ \neg F(c, S_0) \wedge \forall x (x = c \wedge \text{false} \vee x \neq c \wedge F(x, S_0))$$

3 For each case progress the value:

$$\neg F(c, S_1) \wedge \forall x (x = c \wedge \text{true} \vee x \neq c \wedge F(x, S_1)) \quad \vee \\ F(c, S_1) \wedge \forall x (x = c \wedge \text{false} \vee x \neq c \wedge F(x, S_1))$$



Result 2 (strictly local-effect): Example

- KB: $\forall x F(x, S_0)$
- DYN: action $a(x)$ flips the value of $F(x, s)$

Execute action $a(c)$ and progress by modifying the original KB.

1 $a(c)$ may only affect the value of $F(c, S_0)$. Consider cases:

$$F(c, S_0) \wedge KB \quad \vee \quad \neg F(c, S_0) \wedge KB$$

2 For each case unit-propagate “in a first-order way”:

$$F(c, S_0) \wedge \forall x (x = c \wedge \text{true} \vee x \neq c \wedge F(x, S_0)) \quad \vee \\ \neg F(c, S_0) \wedge \forall x (x = c \wedge \text{false} \vee x \neq c \wedge F(x, S_0))$$

3 For each case progress the value:

$$\neg F(c, S_1) \wedge \forall x (x \neq c \supset F(x, S_1))$$



Conclusions

- We investigated the problem of *progression* of basic action theories in the *situation calculus*.
- Under the restriction of local-effects progression is *first-order* (Result 1) and *finite* (Result 2).
- These two results are good news as progression is not first-order representable in the general case.
- (Strictly) local-effect BATs are rich theories that allow:
 - ▶ *incomplete knowledge*: $\exists x \text{key}(x) \wedge \dots$,
($\text{leverUp}(\text{yellow}, S_0) \vee \text{leverUp}(\text{green}, S_0)$).
 - ▶ *infinite domains*: certain things can be *unbounded* instead of insisting on a predefined maximum value.



Current and future work

- Result 2 has been generalized to arbitrary local-effect BATs by Yongmei Liu.
- Getting a finite KB' is not enough. Identify cases where updating is efficient wrt the size of the KB.

