# Lecture 2: Forgetting and Progression

Mikhail Soutchanski

Acknowledgement:
includes examples from the papers “Forget It!" by F.Lin&R.Reiter and
“On First-Order Definability and Computability of Progression for Local-Effect Actions and Beyond" by Y.Liu&G.Lakemeyer

July 10, 2025

## Recap: the Projection Problem

Let *BAT* be a basic action theory, and $Q(s)$ be a situation calculus query formula uniform in $s$. We would like to know whether it is true in $S = do([A_1, \dots, A_n], S_0)$ after executing a sequence of (ground) actions $A_1; \dots; A_n$ starting from $S_0$. We call it the projection problem.

More formally, the *projection problem* is find whether $BAT \models^{?} Query(S)$.

*BAT* does not include the Domain Closure Axiom (DCA) saying that there are finitely many constants $C_1, \dots, C_n$ such that $\forall x(x = C_1 \vee \dots \vee x = C_n)$.
If initial $\mathcal{D}_{S_0}$ has infinitely many constants, *BAT* has infinitely many models.

The initial theory $\mathcal{D}_{S_0}$ does not include the Closed World Assumption (CWA). Therefore, it is an incomplete logical theory. Lecture 3: how to find a plan.

We discussed that the projection problem can be solved using *regression*. Recall that the regression operator distributes inside any logical formulas with an arbitrary main connective $\vee, \wedge, \rightarrow, \forall, \exists$.

When applied to $poss(A_l, S)$, regression replaces $poss(A_l, S)$ with the RHS of the precondition axiom for $A_l$.

When applied to the fluent about $do(A, S)$, regression replaces the fluent with the RHS of its SSA that uses $S$ only. The length of situation decreases by 1.

However, regression can be used only if the length $n$ of the situation term $S$ is finite and does not grow indefinitely. Applying regression recursively becomes computationally inefficient when the situation term is very long.

## Progression: Motivation

Progression is computational mechanism for solving the projection problem. Progression is an alternative to regression.

Regression is basically reasoning backward: from future state back to the initial state. Start with a query about the situation term representing a sequence of actions, and replace the query incrementally with the equivalent formula about an earlier situation. When regression terminates, we got a formula about the initial situation $S_0$ only.

Progression is opposite to regression in the sense that progression is reasoning forward: from current state to the next state. Once an action has been executed, we take a set of formulas describing the initial state, update it accordingly, and obtain a new set of formulas that become true after the executed action.

This operation can be carried out indefinitely as long as we can efficiently update our knowledge after every executed action. Therefore, we have to find classes of actions for which update can be efficiently computed.

Before we study progression, we have to understand **(a)** how old knowledge that is no longer true can be forgotten. Subsequently, we will see **(b)** how new knowledge can be combined into update after forgetting. Progression is done in this order: (a) forget obsolete facts, then (b) include new effects.

## Forgetting: History and an Idea

One can forget about a ground atom or about a predicate in general. We consider the former case. In the simplest case, we can consider forgetting about a propositional atom.

Forgetting was defined by George Boole in the middle of 1850s. About 100 years later, its importance was reaffirmed by Claude Shannon in 1950s, who proposed so-called Shannon expansion for Boolean functions in terms of switching circuits. Later, it found applications in AI thanks to a seminal paper written by Fangzhen Lin and Ray Reiter with the title “Forget It!".

Let $\mathcal{F}(x_1, \dots, x_{k-1}, x_k, x_{k+1}, \dots, x_n)$ be a propositional (Boolean) formula with $n$ propositional variables, and let $\top/\bot$ be propositional constants denoting *True* and *False*. Since each propositional symbol $x_k$ can have only one of the two mutually exclusive truth values, $\mathcal{F}(x_1, \dots, x_{k-1}, x_k, x_{k+1}, \dots, x_n) \equiv$
$x_k \wedge \mathcal{F}(x_1, \dots, x_{k-1}, \top, x_{k+1}, \dots, x_n) \vee \neg x_k \wedge \mathcal{F}(x_1, \dots, x_{k-1}, \bot, x_{k+1}, \dots, x_n)$.

Each of the two sub-expressions does not depend on $x_k$. In this sense, each subexpression forgot about $x_k$. Other than that, each sub-expression keeps all knowledge that was present in the formula $F$ as long as we are concerned only about queries not using $x_k$. Same intuition applies to FOL formulas.

A theory $T'$ is the *result of forgetting* about a ground atom $P(\vec{C})$ in a theory $T$, if $T'$ entails all sentences entailed by $T$ except the ones *related* to $P(\vec{C})$.

## Forgetting: Formal Semantic Definition

We are prepared to consider formal definitions.

### Definition

Let $P(\vec{C})$ be a ground atom, and let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two first-order structures. We write $\mathcal{M}_1 \sim_{P(\vec{C})} \mathcal{M}_2$ and say $\mathcal{M}_1$ and $\mathcal{M}_2$ agree on everything except possibly on the interpretation of $P(\vec{C})$, if:

1. $\mathcal{M}_1$, $\mathcal{M}_2$ have the same domain, and interpret every function identically.
2. For every predicate $Q$ distinct from $P$, $\mathcal{M}_1[Q] = \mathcal{M}_2[Q]$.
3. Let $\vec{u} = \mathcal{M}_1[\vec{C}]$, then for any tuple $\vec{d}$ of the elements in the domain that is distinct from $\vec{u}$, $\vec{d} \in \mathcal{M}_1[P]$ iff $\vec{d} \in \mathcal{M}_2[P]$.

### Definition

Let $T$ be a theory, and $p$ a ground atom. A theory $T'$ is a result of forgetting about $p$ in $T$, denoted by $forget(T,p)$, if for any structure $\mathcal{M}'$, $\mathcal{M}'$ is a model of $T'$ iff there is a model $\mathcal{M}$ of $T$ such that $\mathcal{M} \sim_p \mathcal{M}'$.

**Properties of forgetting.**
If $forget(T;p) = T1$ and $forget(T;p) = T2$, then $T1 \equiv T2$.

$T \models forget(T;p)$, i.e., a theory resulting from forgetting is logically weaker.

## Forgetting: Syntactic Characterization

The result of forgetting a ground atom $P(\vec{C})$ in $\phi$ can be computed by simple syntactic manipulations.

Let $\phi[P(\vec{C})]$ denote the formula obtained from a given input formula $\phi$ by replacing every occurrence of $P(x)$ (if any) in the formula $\phi$ by:

$$(\vec{x} = \vec{C} \land P(\vec{C})) \lor (\vec{x} \neq \vec{C} \land P(\vec{x})).$$

**Property**: $\phi$ and $\phi[P(c)]$ are logically equivalent.
Exercise: prove this statement using Basic Semantic Definition (BSD).

Let $\top/P(c)$ and $\bot/P(c)$ denote substitution of $P(c)$ in $\phi[P(c)]$ with *True* and *False*, respectively. For clarity, we introduce new propositional constant $\top$ as an abbreviation for $(S_0 = S_0)$, and define $\bot$ in the logical language as an abbreviation for $\neg(S_0 = S_0)$. The symbols $\top, \bot$ are more readable.
**Notation.** $\phi^+ = \phi[P(c)](\top/P(c))$ and $\phi^- = \phi[P(c)](\bot/P(c))$.

### Theorem

*(LinReiter1994, Theorem 4) Let $T = \{\phi\}$ be a theory, and $P(\vec{C})$ be a ground atom, then $forget(T, P(\vec{C})) \equiv \phi^+ \lor \phi^-$, where $\phi^+$ and $\phi^-$ are the result of replacing $P(\vec{C})$ by $\top$ and $\bot$ in $\phi[P(\vec{C})]$, respectively.*

Comment: This is saying that formula $forget(T, P(\vec{C}))$ defined semantically can be characterized purely syntactically using sub-formulas $\phi^+, \phi^-$.

## Forgetting: Example 1

(1) Let $\phi$ be formula $\underline{student}(John) \lor \underline{student}(Joe) \lor teacher(John)$. Suppose we would like to forget that $student(John)$, i.e., $p$ is $student(John)$. Then, $\phi[p]$ is obtained by replacing every occurrence of $\underline{student}(x)$ in $\phi$ with the formula

$$x = John \land student(John) \lor x \neq John \land student(x).$$

This yields the following.

$$John = John \land student(John) \lor John \neq John \land student(John) \lor$$
$$John = Joe \land student(John) \lor John \neq Joe \land student(Joe) \lor$$
$$teacher(John).$$

Consequently, $\phi^+_p$ is obtained by replacing $student(John)$ with $\top$

$$John = John \land \top \lor John \neq John \land \top \lor$$
$$John = Joe \land \top \lor John \neq Joe \land student(Joe) \lor$$
$$teacher(John),$$

which is equivalent to $\top$.

It can be similarly shown that $\phi^-_p$ is equivalent to

$$(John \neq Joe \land student(Joe)) \lor teacher(John).$$

Therefore, forgetting $student(John)$ in this formula $\phi$ is

$$forget(\phi, student(John)) \equiv \phi^+_p \lor \phi^-_p \equiv \top.$$

This is intuitively what we expect, since forgetting an atom in any disjunction should be a logical formula weaker than the original disjunction.

## Forgetting: Examples

(2) Let $\phi = \exists x(student(x))$. Notice $\phi[student(John)]$ is an abbreviation for
$\exists x.((x = John \land student(John)) \lor (x \neq John \land student(x)))$.
Then $\phi^+_{student(John)} \equiv \exists x(x = John) \lor \exists x(x \neq John \land student(x))$ which is valid.
Therefore, no matter what $\phi^-_{student(John)}$ is, $forget(\phi; student(John)) = \top$.
Observe after forgetting an atom wrt $\exists$-quantified knowledge, we lost knowledge

(3) Let $\phi = \forall x(student(x))$. By definition of $\phi[P(\vec{C})]$, $\phi[student(John)]$ is
$\forall x.((x = John \land student(John)) \lor (x \neq John \land student(x)))$.

We see that $\phi^+_{student(John)}$ is equivalent to
$\forall x(x = John \lor x \neq John \land student(x))$.

Similarly, $\phi^-_{student(John)}$ is equivalent to
$\forall x(x \neq John \land student(x))$.

Exercise: prove $\forall x(F(x) \lor G(x)) \lor \forall x F(x) \equiv \forall x(F(x) \lor G(x))$ by BSD. So,
$forget(\phi; student(John)) \equiv \forall x(x = John \lor x \neq John \land student(x))$.
But by BSD (do case analysis), since each element of the domain is either John or not John, this is logically equivalent to $\forall x(x \neq John \rightarrow student(x))$.

Observe, our $\forall$-quantified knowledge weakened, but just a bit since we forgot only one atomic fact.

## Forgetting Multiple Atoms at Once

For the purpose of computing progression, when forgetting is applied multiple times, and, possibly to the same predicate symbol more than once, it's possible to define the notion of forgetting about a set of ground atoms at once

Let $G$ be a finite set of ground atoms to be forgotten. A *truth assignment $\mathcal{A}$ to atoms from $G$ is called a $G$-model.* $\mathcal{M}(G)$ denotes the set of all $G$-models.

Let $\phi$ be a formula and $\mathcal{A}$ a $G$-model. Let $P(\vec{c_1})$, $P(\vec{c_2})$, ... $P(\vec{c_m})$ be all those ground atoms in $G$, which use the predicate letter $P$, and let $\mathcal{A}(P(\vec{c_j}))$ represent the truth value of $P(\vec{c_j})$ specified by $\mathcal{A}$.

We use $\phi[\mathcal{A}]$ to denote the result of replacing (for each predicate symbol $P$ in $G$) every occurrence of an atom $P(\vec{x})$ in $\phi$ by the following formula:

$$\bigvee_{j=1}^{m}(\vec{x}=\vec{c_j}\wedge\mathcal{A}[P(\vec{c_j})]) \ \vee\ (\bigwedge_{j=1}^{m}\vec{x}\neq\vec{c_j})\wedge P(\vec{x}). \qquad (1)$$

### Theorem

*(LiuLakemeyer2009, Theorem 2.4) Let $G$ be a finite set of ground atoms and $\phi$ a formula. Then, $forget(\phi, G)$ and $\bigvee_{\mathcal{A}\in\mathcal{M}(G)}\phi[\mathcal{A}]$ are logically equivalent.*

Forgetting about a set $G$ of ground literals amounts to forgetting about each literal one after another, i.e., if $G=\{P_1,\ldots,P_n\}$, then
$$forget(\phi, G) = forget(\cdots forget(forget(\phi, P_1),\ldots,P_n)).$$

## Forgetting multiple atoms at once: Example

Let $\phi=\forall x(clear(x))$ and $G=\{clear(A), clear(B)\}$. Then, by formula (1) we get that $\phi[\mathcal{A}]$ represents
$$\forall x\big(x=A\wedge\mathcal{A}[clear(A)]\vee x=B\wedge\mathcal{A}[clear(B)]\vee(x\neq A\wedge x\neq B\wedge clear(x))\big).$$
For each of the 4 (why 4?) truth assignments $\mathcal{A}$ to atoms in $G$ we compute

$\phi[clear(A)$ is $True$, $clear(B)$ is $True] \equiv$
$\forall x.(x=A\wedge\top\ \vee\ x=B\wedge\top)\vee x\neq A\wedge x\neq B\wedge clear(x)\equiv$
$\forall x.(x=A\vee x=B\vee x\neq A\wedge x\neq B\wedge clear(x))$.

$\phi[clear(A)$ is $True$, $clear(B)$ is $False] \equiv \forall x.(x=A\vee x\neq A\wedge x\neq B\wedge clear(x))$.

$\phi[clear(A)$ is $False$, $clear(B)$ is $True] \equiv \forall x.(x=B\vee x\neq A\wedge x\neq B\wedge clear(x))$.

$\phi[clear(A)$ is $False$, $clear(B)$ is $False] \equiv \forall x.(x\neq A\wedge x\neq B\wedge clear(x))$.

Since the 2nd, 3rd and 4th formulas are special cases of a longer disjunction in the 1st formula, then applying the logical equivalence
$\forall x(F(x)\vee G(x))\vee\forall xF(x)\equiv\forall x(F(x)\vee G(x))$
yields $forget(\phi, G)\equiv\forall x.(x=A\vee x=B\vee x\neq A\wedge x\neq B\wedge clear(x))$.
Exercise: prove (using case analysis in BSD) that the last formula is logically equivalent to $\forall x.(x\neq A\wedge x\neq B\rightarrow clear(x))$

Observe that after forgetting about two atomic facts in $\forall$-quantified sentence, we no longer know the forgotten facts.

## Forgetting Multiple Atoms: Properties

**Properties of forgetting** (exercises to prove).

$forget(forget(T; P1); P2)\equiv forget(forget(T; P2); P1)$

$forget(T1\vee T2; P)\equiv(forget(T1; P)\vee forget(T2; P))$

Proof of Theorem 4 (LiuLakemeyer2009, Theorem 2.4): outline.

$$forget(\phi; G)\equiv forget(\bigvee_{\mathcal{A}\in M(G)}(\phi\wedge\mathcal{A}); G)\equiv\bigvee_{\mathcal{A}\in M(G)}forget(\phi\wedge\mathcal{A}; G)\equiv\bigvee_{\mathcal{A}\in M(G)}\phi[\mathcal{A}]$$

since $\phi[\mathcal{A}]=forget(\phi\wedge\mathcal{A}; G)$.
For example, if $G=P(t)$, then $forget(\phi; P(t))=\phi^{+}\vee\phi^{-}$.

## Progression: Motivation and Formal Definition

Lin and Reiter formalized the notion of progression in AI,v92(1-2)p.131-167. Informally, it involves updating an initial theory $\mathcal{D}_{S_0}$ with the effects of executing an action $\alpha$. The idea: "forget" about all the logical consequences of the fluents in $\mathcal{D}_{S_0}$ that change, and "add" new effects from action $\alpha$. We denote as $S_\alpha$ the situation term $do(\alpha, S_0)$.

### Definition

For two many-sorted structures $\mathcal{M}$, $\mathcal{M}'$ of the situation calculus signature, and a ground action $\alpha$, we write $\mathcal{M}\sim_{S_\alpha}\mathcal{M}'$ if:

1. $\mathcal{M}$ and $\mathcal{M}'$ have the same domains for sorts *action* and *object*;
2. $\mathcal{M}$ and $\mathcal{M}'$ interpret all situation-independent predicate and function symbols identically;
3. $\mathcal{M}$ and $\mathcal{M}'$ agree on interpretation of all fluents at $S_\alpha$, i.e. for every fluent $F$ and every assignment $\sigma$, we have $\mathcal{M},\sigma\models F(\vec{x}, S_\alpha)$ iff $\mathcal{M}',\sigma\models F(\vec{x}, S_\alpha)$.

### Definition

Let $\mathcal{D}$ be a basic action theory with initial theory $\mathcal{D}_{S_0}$ and $\alpha$ be a ground action term. A set of formulas $\mathcal{D}_{S_\alpha}$ (in 2nd-order logic) is called progression of $\mathcal{D}_{S_0}$ to $S_\alpha$ (w.r.t. $\mathcal{D}$ and $\alpha$) if it is uniform in situation term $S_\alpha$ and for any structure $\mathcal{M}$, $\mathcal{M}$ is a model of $\mathcal{D}_{S_\alpha}$ iff there is a model $\mathcal{M}'$ of $\mathcal{D}$ such that $\mathcal{M}\sim_{S_\alpha}\mathcal{M}'$.

## Logical Properties of Progression

Observe that foundational axioms $\Sigma$ are needed if one proofs (usually by induction over $s$) that a state constraint $\forall s C(s)$ is entailed by an action theory. However, to answer projection queries, the smaller set $\mathcal{D} - \Sigma$ is good enough

**Theorem [Lin and Reiter, 1997]** Let $\mathcal{D}$ be a BAT, $\phi(s)$ be a formula uniform in $s$, and **A** be a sequence of ground actions. Then, /* No $\Sigma$ is needed!*/

$$\mathcal{D} \models \phi(do(\mathbf{A}, S_0)) \quad \text{iff} \quad \mathcal{D}_{S_0} \cup \mathcal{D}_{SS} \cup \mathcal{D}_{ap} \cup UNA \models \phi(do(\mathbf{A}, S_0)).$$

This can be proved by induction over structure of $\phi$ from the following.

**Proposition 3.2 [Lin and Reiter, 1997**] Given any model $\mathcal{M}^-$ of $\mathcal{D} - \Sigma$, there is a model $\mathcal{M}$ of $\mathcal{D}$ such that:
1. $\mathcal{M}^-$ and $\mathcal{M}$ have the same domains for sorts action and object, and interpret all situation independent predicates and functions identically,
2. for any sequence **A** of ground action terms, any fluent $F$, and any variable assignment $\nu$: $\mathcal{M}, \nu \models F(\bar{x}, do(\mathbf{A}, S_0))$ iff $\mathcal{M}^-, \nu \models F(\bar{x}, do(\mathbf{A}, S_0))$.

**Proposition 4.3 [Lin and Reiter, 1997**] Let $\mathcal{D}_{S_\alpha}$ be a progression of the initial $\mathcal{D}_{S_0}$ to $S_\alpha$. Every model of $\mathcal{D}$ is a model of $\Sigma \cup \mathcal{D}_{SS} \cup \mathcal{D}_{ap} \cup UNA \cup \mathcal{D}_{S_\alpha}$.

Proposition 4.4 shows that for any model of progressed BAT, there exists a model of original BAT such that they agree on all fluents in future of $S_\alpha$.

**Theorem 4.5 [Lin and Reiter, 1997**] Let $\mathcal{D}_{S_\alpha}$ be a progression of $\mathcal{D}_{S_0}$ to $S_\alpha$. For any (possibly 2nd order) sentence $\phi$ uniform in $S_\alpha$, $\mathcal{D}_{S_\alpha} \models \phi$ iff $\mathcal{D} \models \phi$.

This important theorem informs us that $\mathcal{D}_{S_\alpha}$ is a *strongest postcondition* of the precondition $\mathcal{D}_{S_0}$, w.r.t. action $\alpha$. This is a key to tractability, if $\mathcal{D}_{S_\alpha}$ is in FO.

## Progression: When it Can Be Efficiently Computed?

For *strictly context free* SSAs, if an initial $\mathcal{D}_{S_0}$ is a (possibly infinite) set of literals, then progression $\mathcal{D}_{S_\alpha}$ is merely an insertion and removal of literals based on the effects of $\alpha$. In this important special case, progression $\mathcal{D}_{S_\alpha}$ is a formula in FOL: see Section 8.1 in [Lin and Reiter, 1997]. This generalizes STRIPS, where $\mathcal{D}_{S_0}$ is a finite set of atoms, and both DCA and CWA apply.

For a general $\mathcal{D}_{S_0}$, progression is always definable as a 2nd order theory.

It turns out, first order definable progression is too weak: see an example in Vassos& Levesque "How to progress a database III", AI, 2013,v195, p203-221

We can be interested in a middle point between these extremes. Liu&Lakemeyer prove in their paper (Theorem 3.6) that for a local-effect BAT, if an initial $\mathcal{D}_{S_0}$ is in *proper+*, then its progression remains in *proper+* (in FO).

We start with a discussion of examples of local and global effect actions.

An action has a <u>*local*</u> effect on a fluent, if all arguments of the fluent are subsumed by the arguments of the action. Action's arguments determine all changes that may happen. Since each action has finitely many arguments, in this case there might be a few local changes only.

Informally, an action $A$ has a <u>*non-local*</u> effect on a fluent, if other objects (not mentioned in $A$) can be affected by an action. This happens if at least 1 argument of the fluent is not mentioned in the arguments of the action $A$.

## Successor State Axioms for Blocks World

Fluents: $On(x, y)$ and $Ontable(x)$
Actions: $moveToTable(x)$ and $move(x, y)$
Below, we assume that all free variables $x, y, a, s$ are $\forall$-quantified at front.

$$On(x, y, do(a, s)) \leftrightarrow \quad a = move(x, y) \lor$$
$$On(x, y, s) \land a \neq moveToTable(x) \land \neg\exists z(a = move(x, z)).$$

Action $move(x, y)$ has a positive local effect
Action $moveToTable(x)$ has a negative non-local effect

$$Ontable(x, do(a, s)) \leftrightarrow \quad a = moveToTable(x) \lor$$
$$Ontable(x, s) \land \neg\exists y(a = move(x, y)).$$

Actions $moveToTable(x)$ and $move(x, y)$ have local effect

$$Clear(x, do(a, s)) \leftrightarrow \quad \exists y, z(a = move(y, z) \land On(y, x, s)) \lor$$
$$\exists y(a = moveToTable(y) \land On(y, x, s)) \lor$$
$$Clear(x, s) \land \neg\exists w(a = move(w, x)).$$

Actions $move(y, z)$ and $moveTotable(y)$ have non-local effects

Sometimes, an action has an effect only on a few objects, but it is no local according to the definitions. In this case, we say action has a non-local effect.

Subsequently, we consider another version of the blocks world, where all actions have only local effects.

## Successor State Axioms for Logistics

Fluents: $at(x, loc, s)$, $x$ is at a location $loc$ in situation $s$.
Actions: $drive(t, loc1, loc2, city)$ moves the truck $t$ from $loc1$ to $loc2$ and is possible only if the truck $t$ is at $loc1$ and both locations are in the same $city$.
$fly(air, locFrom, locTo)$ is possible if $air$ is an airplane at $locFrom$, and if both locations are airports in distinct cities.

$$\forall x, y, a, s.\ at(x, y, do(a, s)) \leftrightarrow$$
$$\exists z_1, \exists z_2 \big(a = drive(x, z_1, y, z_2) \land city(z_2) \land truck(x) \land loc(z_1) \land$$
$$in\_city(z_1, z_2) \land loc(y) \land in\_city(y, z_2)\big) \lor$$
$$\exists z_1, \exists z_2, \exists z_3 \big(a = drive(z_1, z_2, y, z_3) \land obj(x) \land truck(z_1) \land$$
$$loaded(x, z_1, s) \land loc(z_2) \land city(z_3) \land in\_city(z_2, z_3) \land$$
$$loc(y) \land in\_city(y, z_3) \land (y \neq z_2)\big) \lor$$
$$\exists z_1\ (a = fly(x, z_1, y) \land airplane(x) \land airport(z_1) \land airport(y)) \lor$$
$$\exists z_1, \exists z_2\ (a = fly(z_1, z_2, y) \land obj(x) \land airplane(z_1) \land airport(z_2) \land$$
$$loaded(x, z_1, s) \land airport(y)) \lor$$
$$at(x, y, s) \land \ldots \text{/* unless the last action makes this fluent false */} \ldots$$

The first $drive(x, z_1, y, z_2)$ and the first $fly(x, z_1, y)$ actions have local effects on $at(x, y, do(a, s))$ since both object arguments of the fluent occur as arguments of these actions. The second $drive(z_1, z_2, y, z_3)$ and the second $fly(z_1, z_2, y)$ actions have global effects on this fluent, since location of all objects inside a vehicle changes when the vehicle moves to destination.

## Local-effect Actions and BATs: Definition

$$(\forall \vec{x} \forall s \forall a).\ \ F(\vec{x}, do(a,s)) \leftrightarrow \gamma_F^+(\vec{x}, a,s) \lor F(\vec{x}, s) \land \neg\gamma_F^-(\vec{x}, a,s), \qquad (2)$$

where $\vec{x}$ is a tuple of object arguments of the fluent $F$, and each of the $\gamma_F$'s is a disjunction of uniform formulas
$[\exists \vec{z}].a = A(\vec{u}) \land \phi(\vec{x}, \vec{z}, s)$, /* a set of variables $\vec{z} \subseteq \vec{u}$; may be $\vec{x} \subset \vec{u}$ */
where $A(\vec{u})$ is an action with a tuple $\vec{u}$ of object arguments, $\vec{z}$ are optional extra arguments, and $\phi(\vec{x}, \vec{z}, s)$ is a context condition.

A set of variables $\vec{z}$ in a context condition $\phi(\vec{x}, \vec{z}, s)$ must be a subset of object variables $\vec{u}$. If $\vec{u}$ in an action function $A(\vec{u})$ does not include any $z$ variables, then there is no $\exists\vec{z}$ quantifier.

If not all variables from $\vec{x}$ are included in $\vec{u}$, then it is said that $A(\vec{u})$ has a *non-local effect*. The fluent $F$ has at least one $\forall$-quantified object argument $x$ not included in $\vec{u}$, but it is constrained by $\phi(\vec{x}, \vec{z}, s)$. $F$ experiences changes beyond the objects explicitly named in $A(\vec{u})$.

When a vector of object variables $\vec{u}$ contains both $\vec{x}$ and $\vec{z}$, we say that the action $A(\vec{u})$ has a *local effect*.

A BAT is called a *local-effect BAT* if all of its actions have only local effects. In a local-effect action theory, each action can change values of fluents only for objects explicitly named as arguments of the action. How do we know this?

## Local effect SSAs with quantifier-free contexts

In a local-effect SSA, consider substitution of a ground action term $A(\vec{B_x}, \vec{B_z})$ for a variable $a$ in the formula $[\exists \vec{z}].a = A(\vec{x}, \vec{z}) \land \phi(\vec{x}, \vec{z}, s)$.

Applying UNA for action terms yields $[\exists \vec{z}].\vec{x} = \vec{B_x} \land \vec{z} = \vec{B_z} \land \phi(\vec{x}, \vec{z}, s)$. Next, applying FOL equivalence $\exists z(z = B \land \phi(z)) \equiv \phi(B)$ repeatedly for each $z_i$ results in the logically equivalent formula $\vec{x} = \vec{B_x} \land \phi(\vec{x}, \vec{B_z}, s)$. We make the following realistic simplifying assumption.

### Definition

An SSA is essentially quantifier-free if for each ground action $\alpha$, each context condition $\phi(\vec{x}, \vec{z}, s)$ is a quantifier-free formula, after using *UNA* for actions, and $\phi(\vec{x}, \vec{z}, s)$ consists of fluents.

In other words, each of $\gamma_F^+(x, a, s)$ and $\gamma_F^-(x, a, s)$ can be simplified to a boolean combination of equalities, inequalities, and fluents.

Example: the SSA for BW are essentially quantifier-free; context conditions are single fluent.

Most of the examples of SSAs in the planning benchmarks satisfy this additional condition. Therefore, this is a reasonable syntactic restriction on the basic action theories.

## Example: A simplified BW with local-effects

There is a single action, $move(x, y, z)$, moving a block $x$ from block $y$ to block $z$ that can be the table. We consider 3 fluents.
$clear(x, s)$, block $x$ has no blocks on top of it;
$on(x, y, s)$, block $x$ is on block $y$ in situation $s$;
$eh(x, s)$, the height of block $x$ is even in the tower of blocks below $x$.

The following SSAs are local-effect (with implicit $\forall x, y, a, s$ at front):

$$clear(x, do(a, s)) \leftrightarrow \quad (\exists y, z)a = move(y, x, z) \lor clear(x, s) \land \neg(\exists y, z)a = move(y, z, x),$$

$$on(x, y, do(a, s)) \leftrightarrow \quad (\exists z)a = move(x, z, y) \lor on(x, y, s) \land \neg(\exists z)a = move(x, y, z),$$

$$eh(x, do(a, s)) \leftrightarrow \quad (\exists y, z)[a = move(x, y, z) \land \neg eh(z, s)] \lor$$
$$eh(x, s) \land \neg(\exists y, z)[a = move(x, y, z) \land eh(z, s)].$$

Instantiate *a* with a ground action $\alpha = move(C_1, C_2, C_3)$. What would we get?

$$clear(x, do(move(C_1,C_2,C_3), s)) \leftrightarrow \quad (\exists y, z)move(C_1,C_2,C_3) = move(y, x, z) \lor$$
$$clear(x, s) \land \neg(\exists y, z)move(C_1,C_2,C_3) = move(y, z, x),$$

$$on(x, y, do(move(C_1,C_2,C_3), s)) \leftrightarrow \quad (\exists z)move(C_1,C_2,C_3) = move(x, z, y) \lor$$
$$on(x, y, s) \land \neg(\exists z)move(C_1,C_2,C_3) = move(x, y, z),$$

$$eh(x, do(move(C_1,C_2,C_3), s)) \leftrightarrow (\exists y, z)[move(C_1,C_2,C_3) = move(x, y, z) \land \neg eh(z, s)] \lor$$
$$eh(x, s) \land \neg(\exists y, z)[move(C_1,C_2,C_3) = move(x, y, z) \land eh(z, s)].$$

Applying UNA for actions to $(\exists y, z)move(C_1,C_2,C_3) = move(y, x, z)$ yields the formula $\exists y, z(y = C_1 \land x = C_2 \land z = C_3)$ which is logically equivalent to (Why?)
$\big(x = C_2 \land \exists y(y = C_1) \land \exists z(z = C_3)\big) \equiv x = C_2$.

## Transformed SSA, Argument set, Characteristic set

Applying UNA for actions and doing simplifications yields *transformed* SSAs.

$$clear(x, do(move(C_1,C_2,C_3), s)) \leftrightarrow \quad (x = C_2) \lor clear(x, s) \land \neg(x = C_3),$$

$$on(x, y, do(move(C_1,C_2,C_3), s)) \leftrightarrow \quad (x = C_1 \land y = C_3) \lor on(x, y, s) \land \neg(x = C_1 \land y = C_2),$$

$$eh(x, do(move(C_1,C_2,C_3), s)) \leftrightarrow \quad (x = C_1) \land \neg eh(C_3, s) \lor \ eh(x, s) \land \neg(x = C_1 \land eh(C_3, s$$

The *argument set* $\Delta_F$ (read as "Delta") for the fluent $F$ wrt a ground action $\alpha$ is a <u>set of constants</u> appearing in the transformed SSA for $F$ instantiated with $\alpha$.

The set $\Delta_{clear}$ for the fluent $clear(x, s)$ wrt $move(C_1,C_2,C_3)$ is $\{C_2, C_3\}$. For the fluent $on(x, y, s)$, $\Delta_{on} = \{\langle C_1,C_3\rangle, \langle C_1,C_2\rangle\}$. For $eh(x, s)$, $\Delta_{eh} = \{C_1\}$.

The *characteristic set* $\Omega$ (read as "Omega") of a ground action is a set of all ground atoms subject to change by this action. E.g., for $move(C_1, C_2, C_3)$

$$\Omega(s) = \{clear(C_2, s), clear(C_3, s), on(C_1, C_3, s), on(C_1, C_2, s), eh(C_1, s)\}.$$

If block $C_3$ is clear at $s$, it no longer remains clear after doing $move(C_1,C_2,C_3)$, but block $C_2$ will become clear.

Use $\Delta_F$ to instantiate the transformed SSA for $F$: replace object arguments of $F$ with constants from $\Delta_F$. Obtain the set $\mathcal{D}_{ss}[\Omega]$ of formulas representing <u>new</u> values of fluents, e.g., $on(C_1,C_3, do(move(C_1,C_2,C_3), S_0)) \leftrightarrow$

$$C_1 = C_1 \land C_3 = C_3 \lor on(C_1,C_3,S_0) \land \neg(C2 = C_3 \land C_3 = C_2).$$

The set $\mathcal{D}_{ss}[\Omega]$ of instantiated SSAs wrt $\Omega(S_\alpha)$ is the following : $\{clear(C_2,S_\alpha),$
$\neg clear(C_3,S_\alpha), on(C_1,C_3,S_\alpha), \neg on(C_1,C_2,S_\alpha), eh(C_1,S_\alpha) \leftrightarrow \neg eh(C_3,S_0)\}$.

## Progression for a Local-effect BAT: General Case

For each fluent $F$, collect constants where $F$ changes into the argument set $\Delta_F$. Progression involves (a) forgetting about all those affected ground fluents in the initial theory, and (b) computing their new values from SSAs.

The set of these ground fluents is called the characteristic set of $\alpha$ and is denoted by $\Omega(s)$. The set $\Omega(S_0)$ is all ground fluents to be forgotten.

Let $\mathcal{D}_{SS}[\Omega]$ denote the instantiation of all SSAs w.r.t. $\Omega(S_0)$, i.e. the set of

$$F(\vec{t}, S_\alpha) \equiv \Phi_F(\vec{t}, \alpha, S_0),$$

where $F(\vec{t}, S_\alpha) \in \Omega(S_\alpha)$ and $\Phi_F$ is the instantiated right hand-side of the SSA for fluent $F$ w.r.t. a ground action $\alpha$.

### Theorem

*[Liu&Lakemeyer2009, Theor. 3.6] Let $\mathcal{D}$ be a local-effect BAT, $\alpha$ a ground action, and $\Omega(s)$ be the characteristic set of $\alpha$. Let $\phi$ be $\mathcal{D}_{S_0} \wedge \mathcal{D}_{SS}[\Omega]$. Then the following is a progression $\mathcal{D}_{S_\alpha}$ of $\mathcal{D}_{S_0}$ w.r.t. $\alpha$:*

$$\mathcal{D}_{S_\alpha} = \bigwedge UNA \wedge forget(\phi, \Omega(S_0))\,(S_\alpha/S_0),$$

*where $\psi(u/v)$ is the result of replacing all occurrences of $v$ in $\psi$ by $u$. Moreover, using notation from Theorem about forgetting we have:*

$$forget(\phi, \Omega(S_0)) = \bigvee_{\mathcal{A} \in \mathcal{M}(\Omega(S_0))} \phi[\mathcal{A}] \;\; (S_\alpha/S_0).$$

This computation can be simplified using *irrelevance*, which divides formulas into those which are affected, or not affected by forgetting about $P(\vec{C})$.

## References

Gerhard Lakemeyer and H.J. Levesque: “Evaluation-Based Reasoning with Disjunctive Information in First-Order Knowledge Bases". KR 2002: pages 73-81 (defines *proper+* KBs).

Hector J. Levesque: “A Completeness Result for Reasoning with Incomplete First-Order Knowledge Bases". KR 1998: pages 14-23 (defines *proper* KBs).

Fangzhen Lin and Raymond Reiter: "Forget It!", 1994 AAAI Fall Symposium, Technical Report FS-94-02.

Fangzhen Lin, Raymond Reiter: “How to Progress a Database". Artif. Intell. 1997, vol 92(1-2): pages 131-167.

Yongmei Liu, Gerhard Lakemeyer: “On First-Order Definability and Computability of Progression for Local-Effect Actions and Beyond". IJCAI 2009: pages 860-866.

Stavros Vassos, Hector J. Levesque: “How to progress a database III". Artif. Intell. 2013, vol. 195, pages 203-221

## Irrelevance

**Definition** Let $P(\vec{C})$ be an atom, $\phi$ be a sentence. $P(\vec{C})$ is *irrelevant* to $\phi$ iff

$$forget(\phi, P(\vec{C})) \equiv \phi.$$

Note that when $\phi$ has no occurrences of $P(\vec{C})$, then $P(\vec{C})$ is irrelevant to $\phi$.

For simplicity, let us assume that both an initial theory $\mathcal{D}_{S_0}$ and formulas in $\mathcal{D}_{SS}[\Omega]$ are a knowledge base (KB) which is a conjunction of clauses, where each clause is a disjunction of ground literals (called *proper+* KB).

To simplify the task of forgetting about a ground atom $p$, we will rearrange clauses as follows. Using the distributivity law $((a \vee p) \wedge (b \vee p)) \equiv (a \wedge b \vee p)$ one can collect sub-formulas from all clauses in a KB with positive occurrences of $p$ into a single conjunction $\phi_{pos}$. Similarly, using the law $((a \vee \neg p) \wedge (b \vee \neg p)) \equiv (a \wedge b \vee \neg p)$ one can combine sub-formulas occurring together with $\neg p$ in clauses of a KB into the conjunction $\phi_{neg}$.

**Proposition** [(Liu&Lakemeyer2009, Prop. 5.3]
Let $P(\vec{C})$ be a ground atom, $\phi_{pos}$, $\phi_{neg}$, and $\phi_{irr}$ be sentences to which $P(\vec{C})$ is irrelevant, and $KB$ be $(P(\vec{C}) \vee \phi_{pos}) \wedge (\neg P(\vec{C}) \vee \phi_{neg}) \wedge \phi_{irr}$. Then, $forget(KB, P(\vec{C})) = (\phi_{pos} \wedge \phi_{irr}) \vee (\phi_{neg} \wedge \phi_{irr}) \equiv (\phi_{pos} \vee \phi_{neg}) \wedge \phi_{irr}$.

Examples. $forget(P(\vec{C}) \wedge \phi_{irr}, P(\vec{C})) = \phi_{irr}$. $forget(\neg P(\vec{C}) \wedge \phi_{irr}, P(\vec{C})) = \phi_{irr}$.

Since $\Omega(S_0)$ is a **set** of ground fluent atoms, we apply similar transformations to $\mathcal{D}_{S_0} \cup \mathcal{D}_{SS}[\Omega]$ repeatedly to compute progression $\mathcal{D}_{S_\alpha}$ linear wrt initial $\mathcal{D}_{S_0}$.