Progression and Regression Using Sensors

Giuseppe De Giacomo
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Rome, Italy
degiacomo@dis.uniroma1.it

Hector Levesque
Department of Computer Science
University of Toronto
Toronto, Canada M5S 3H5
hector@cs.toronto.edu

Abstract

In this paper, we consider the projection task (determining what does or does not hold after performing a sequence of actions) in a general setting where a solution to the frame problem may or may not be available, and where online information from sensors may or may not be applicable. We formally characterize the projection task for actions theories of this sort, and show how a generalized form of regression produces correct answers whenever it can be used. We characterize conditions on action theories, sequences of actions, and sensing information that are sufficient to guarantee that regression can be used, and present a provably correct regression-based procedure in Prolog for performing the task under these conditions.

1 Introduction

One of the most fundamental tasks concerned with reasoning about action and change is the *projection task*: determining whether a fluent[1] does or does not hold after performing a sequence of actions. In the usual formulation, we are given a characterization of the initial state of the world and a specification of some sort of what each action does. The projection task requires us to determine the cumulative effects (and non-effects) of sequences of actions.

Projection is clearly a prerequisite to *planning*: we cannot figure out if a given goal is achieved by a sequence of actions if we cannot determine what holds after doing the sequence. Similarly, the *high-level program execution task* [6], which is that of finding a sequence of actions constituting a legal execution of a high-level program, also requires projection: to execute a program like "while there is a block on the table, pick up a block and put it away," one needs to be able to determine after various sequences of actions if there is still a block on the table.

A perennial stumbling block in the specification of the projection task is the frame problem [4]: for each action, we need to specify somehow not only what changes as the result of performing the action, but the much larger number of fluents unaffected by the action. One solution to this difficulty is make a STRIPS assumption [2]: what will be known about a state of the world will be representable as a database of simple atomic facts, and we specify actions as operators on such a database, adding or removing just what changes.

[1]By a fluent, we mean a property of the world that changes as the result of performing actions.

A much more expressive and declarative solution to the frame problem is presented in [10]. There, the situation calculus is used to specify the effects of actions, and then a simple syntactic procedure is provided for combining the effects for each fluent into a so-called *successor state axiom* that logically entails not only the effect axioms, but all the frame axioms for that fluent as well.

However, this solution to the frame problem makes a strong completeness assumption: after specifying the (perhaps conditional) effects of the given actions on fluents, and then allowing for possible ramifications of these actions (*e.g.* [7]), it is then assumed that a fluent changes *only if* it has been affected in one of these ways. Thus, it is assumed that each fluent can be *regressed* in the sense that whether or not it holds after performing an action can be determined by considering the action in question and what was true just before.

What is not allowed, in other words, are cases where the value of a fluent does not depend in this way on the previous state. This can arise in at least two ways. First, a fluent might change as the result of an action that is exogenous to the system. If a robot opens a door in a building, then when nobody else is around, it is justified in concluding that the door remains open until the robot closes it. But in a building with other occupants, doors will be opened and closed unpredictably. Similarly, the robot may be able to determine that a warning light is on simply because it was on in the previous state and the robot is the only one who can turn it off; but it may not be able to predict when the warning light goes on. Secondly, the robot might have incomplete knowledge of the fluent in question. For example, a robot normally would not be able to infer the current temperature outdoors, since this is the result of a large number of unknown events and properties. Even when a fluent is expected to stay relatively constant, like the depth of water in a swimming pool, the robot may not know what that value is.

In cases such as these, the only way we can expect a robot to be able to perform the projection task is if it has some other way of determining the current value of certain fluents in the world. To this effect, we assume that not only can the robot use regression, it can use a collection of onboard *sensors*. In [5], sensing is modeled as an action performed by a robot that returns a binary measurement. The robot then uses so-called *sensed fluent axioms* to correlate the value returned with the state of various fluents. However, in this account, no attempt is made to be precise about the exact relation between sensing and regression. Moreover, there is no possibility of saying that only under certain conditions can regression be used, and in others, sensing.

What we propose in this paper is this: a formal specification of a changing world that generalizes Reiter's solution to the frame problem (and hence STRIPS also) to allow conditional successor state axioms, and generalizes Levesque and others' treatment of sensors (*e.g.* [1; 3; 9; 12]) to allow conditional sensing axioms. Our specification will be sufficiently general that in some cases, there will simply not be enough information to perform the projection task even with sensing. However, in many cases, we will be able to do projection using a combination of sensing and regression. In addition to this specification, we propose a reasoning method for performing projection under these general circumstances which is guaranteed to be sound, and in many cases of interest, complete. We provide a Prolog evaluation procedure for the projection task and prove its soundness and under suitable circumstances its completeness.

2 Basic action theories

Our account of action, sensing, and change is formulated in the language of the situation calculus [4; 11]. We will not go over the language here except to note the following components: there is a special constant S_0 used to denote the *initial situation*, namely the one in which no actions have yet occurred; there is a distinguished binary function symbol *do* where $do(a, s)$ denotes the successor situation to s resulting from performing action a; relations whose truth values vary from situation to situation, are called (relational) *fluents*, and are denoted by predicate symbols taking a situation term as their last argument; and there is a special predicate $Poss(a, s)$ used to state that action a is executable in situation s.

Within this language, we can formulate action theories that describe how the world changes as the result of the available actions. One such is a theory of the following form [10]:

- Axioms describing the initial situation S_0, and axioms not mentioning situations at all, which form together the initial database.
- Action precondition axioms, one for each primitive action a, characterizing $Poss(a, s)$.
- Successor state axioms, one for each fluent F, stating under what conditions $F(\vec{x}, do(a, s))$ holds as function of what holds in situation s. These take the place of the so-called effect axioms, but also provide a solution to the frame problem [10].
- Unique names axioms for the primitive actions.
- Some foundational, domain independent axioms.

For example, the successor state axiom[2]

$$\begin{aligned} &Broken(x, do(a, s)) \equiv \\ &\quad a = drop(x) \wedge Fragile(x) \\ &\vee\ \exists b\, [a = explode(b) \wedge Bomb(b) \wedge Near(x, b, s)] \\ &\vee\ a \neq repair(x) \wedge Broken(x, s) \end{aligned}$$

[2]Here and below, formulas should be read as universally quantified from the outside.

states that an object x is broken after doing action a if a is dropping it and x is fragile, a is exploding a bomb near it, or it was already broken, and a is not the action of repairing it.

In [5], to characterize the result of sensing, it is assumed that each primitive action can return a binary sensing result, and that there is a special predicate $SF(a, s)$ used to state that action a returns value 1 in situation s. To relate this sensing result to fluents, the following are added to basic action theories:

- Sensed fluent axioms, one for each primitive action a, characterizing SF.

For example, the sensed fluent axiom

$$SF(readHeatGauge, s) \equiv \exists n.\ RobotTemp(n, s) \wedge n > 25$$

states that reading the heat gauge returns 1 iff the temperature around the robot exceeds 25 degrees.

3 Guarded action theories

In what follows we will be replacing successor state and sensed fluent axioms by more general versions. To this effect, instead of assuming that actions return a binary sensing value, we assume that a robot has a number of onboard sensors that provide sensing readings at any time. Thus, we drop SF from the language of the situation calculus, and introduce instead a finite number of *sensing functions*: unary functions whose only argument is a situation. For example, $\mathsf{thermometer}(s)$, $\mathsf{sonar}(s)$, $\mathsf{depthGauge}(s)$, might all be real-valued sensing functions.[3]

We then define a *sensor-fluent formula* to be a formula of the language (without *Poss*, for simplicity) that uses at most a single situation term, which is a variable, and that this term only appears as the final argument of a fluent or sensor function. We write $\phi(\vec{x}, s)$ when ϕ is a sensor-fluent formula with free variables among the $\vec{x}$ and s, and $\phi(\vec{t}, t_s)$ for the formula that results after the substitution of $\vec{x}$ by the vector of terms $\vec{t}$ and s by the situation term t_s. A *sensor formula* is a sensor-fluent formula that mentions no fluents, and a *fluent formula* is one that mentions no sensor functions.

We then define our generalized version of successor state and sensed fluent axioms as follows:

> A *guarded successor state axiom* (GSSA) is a formula of the form
>
> $$\alpha(\vec{x}, a, s) \supset [F(\vec{x}, do(a, s)) \equiv \gamma(\vec{x}, a, s)]$$
>
> and a *guarded sensed fluent axiom* (GSFA) is a formula of the form
>
> $$\alpha(\vec{x}, s) \supset [F(\vec{x}, s) \equiv \rho(\vec{x}, s)]$$
>
> where α is a sensor-fluent formula called the *guard* of the axiom, F is a relational fluent, γ is a fluent formula, and ρ is a sensor formula.

An action theory can contain any number of GSSAs and GSFAs for each fluent. We can handle a universally applicable successor state axiom like the one for *Broken* above by using the guard **True**. We no longer have sensing actions, but we

[3]Syntactically, these look like functional fluents, so to avoid confusion, we only deal with relational fluents in this paper.

can achieve much the same effect using a GSFA with guard **True**. For example,

$$\textbf{True} \supset [\mathit{RobotTemp}(n,s) \equiv \mathsf{thermometer}(s) = n]$$

says that the on board thermometer always measures the temperature around the robot.

3.1 Some examples

We now proceed to consider examples that cannot be represented in the basic action theories from Section 2.

1. The outdoor temperature is unpredictable from state to state. However, when the robot is outdoors, its onboard thermometer measures that temperature.

$$\begin{aligned}&\mathit{Outdoors}(s) \supset\\ &\quad \mathit{OutdoorTemp}(n,s) \equiv \mathsf{thermometer}(s) = n\end{aligned}$$

   Note that when the guard is false, *i.e.* when the robot is indoors, nothing can be concluded regarding the outdoor temperature.

2. The indoor temperature is constant when the climate control is active, and otherwise unpredictable. However, when the robot is indoors, its onboard thermometer measures that temperature:

$$\begin{aligned}&\mathit{Indoors}(s) \supset\\ &\quad \mathit{IndoorTemp}(n,s) \equiv \mathsf{thermometer}(s) = n\end{aligned}$$

$$\begin{aligned}&\mathit{ClimateControl}(s) \supset\\ &\quad \mathit{IndoorTemp}(n, do(a,s)) \equiv \mathit{IndoorTemp}(n,s)\end{aligned}$$

   Note that in this case, if the climate control remains active, then a robot that goes first indoors and then outdoors will still be able to infer the current indoor temperature using both sensing and regressing. To our knowledge, no other representation for reasoning about action and change can accommodate this combination.

3. The distance between a (1-dimensional) robot and the wall is affected only the moving actions. Also, the onboard sonar correctly measures the distance to the wall, but only when the reading is within a certain interval.

$$\begin{aligned}&\textbf{True} \supset\\ &\quad \mathit{WDist}(n, do(a,s)) \equiv\\ &\qquad\quad a = \mathit{forward} \wedge \mathit{WDist}(n+1,s)\\ &\qquad \vee\ a = \mathit{backward} \wedge \mathit{WDist}(n-1,s)\\ &\qquad \vee\ a \neq \mathit{forward} \wedge a \neq \mathit{backward} \wedge \mathit{WDist}(n,s)\end{aligned}$$

$$\begin{aligned}&\theta_1 \leq \mathsf{sonar}(s) \leq \theta_2 \supset\\ &\quad \mathit{WDist}(n,s) \equiv \mathsf{sonar}(s) = n\end{aligned}$$

   In this case, the successor state axiom is universally applicable, meaning we can always regress all the way to S_0 to determine the distance to the wall. However, if the distance to the wall in S_0 is unknown, we would still not know the current value, and so it much more useful to be able to regress to a situation where the sonar reading was within its operating range.

4. If the robot is alone in the building, the state of the door is completely determined by the robot's *open* and *close* actions. Either way, any time the robot is in front of the door, its onboard door sensor correctly determines the state of the door.

$$\begin{aligned}&\mathit{Alone}(s) \supset\\ &\quad \mathit{DoorOpen}(x, do(a,s)) \equiv\\ &\qquad\quad a = \mathit{open}(x)\\ &\qquad \vee\ a \neq \mathit{close}(x) \wedge \mathit{DoorOpen}(x,s)\end{aligned}$$

$$\begin{aligned}&\mathit{InFrontOf}(x,s) \supset\\ &\quad \mathit{DoorOpen}(x,s) \equiv \mathsf{doorSensor}(s) = 1\end{aligned}$$

   One intriguing possibility offered by this example is that on closing a door, and later coming back in front of the door to find it open, a security guard robot would be able to infer that $\neg\mathit{Alone}$.

5. A warning light for an alarm can go on unpredictably. Once it is on, however, it will stay on until the robot turns it off. Also, the robot can determine the state of the warning using its onboard light sensor, provided it is looking at the light.

$$\begin{aligned}&\mathit{LookingAt}(x,s) \supset\\ &\quad \mathit{LightOn}(x,s) \equiv \mathsf{lightSensor}(s) = 1\end{aligned}$$

$$\begin{aligned}&\mathit{LightOn}(x,s) \vee a = \mathit{turnoff}(x) \supset\\ &\quad \mathit{LightOn}(x, do(a,s)) \equiv\\ &\qquad a \neq \mathit{turnoff}(x) \wedge \mathit{LightOn}(x,s)\end{aligned}$$

   In this case, we need a complex guard for the successor state axiom, since we can only regress when the light was on previously or when the action is to turn it off.

3.2 Histories and the projection task

We are now ready to define the projection task formally. Obviously, to be able to determine if a fluent holds at some point, it is no longer sufficient to know just the actions that have occurred; we also need to know the readings of the sensors along the way. Consequently, we define a *history* as a sequence of the form $(\vec{\mu_0}) \cdot (A_1, \vec{\mu_1}) \cdots (A_n, \vec{\mu_n})$ where A_i $(1 \leq i \leq n)$ is a ground action term and $\vec{\mu_i} = \langle \mu_{i1}, \ldots, \mu_{im} \rangle$ $(0 \leq i \leq n)$ is a vector of values, with μ_{ij} understood as the reading of the j-th sensor after the i-th action. If λ is such a history, we then recursively define a ground situation term $\mathit{end}[\lambda]$ by $\mathit{end}[(\vec{\mu_0})] = S_0$ and $\mathit{end}[\lambda \cdot (A, \vec{\mu})] = do(A,t)$ where $t = \mathit{end}[\lambda]$. We also define a ground sensor formula $\mathit{Sensed}[\lambda]$ as $\bigwedge_{i=0}^{n} \bigwedge_{j=1}^{m} h_j(\mathit{end}[\lambda_i]) = \mu_{ij}$ where λ_i is the subhistory up to action i, $(\vec{\mu_0}) \cdots (A_i, \vec{\mu_i})$, and h_j is the j-th sensor function. So $\mathit{end}[\lambda]$ is the situation that results from doing the actions in λ and $\mathit{Sensed}[\lambda]$ is the formula that states that the sensors had the values specified by λ [4]. The projection task, then, is this:

> Given an action theory Σ as above, a history λ, and a formula $\phi(s)$ with a single free variable s, determine whether or not $\Sigma \cup \mathit{Sensed}[\lambda] \models \phi(\mathit{end}[\lambda])$.

4 Generalized regression

In principle, the projection task as formulated can be solved using a general first-order theorem-prover. But the ineffectiveness of this approach in an even simpler setting is arguably what led many to abandon the situation calculus and

[4]Obviously interesting histories λ have to satisfy certain legality criteria such as consistency of $\Sigma \cup \mathit{Sensed}[\lambda]$ and conformance to *Poss*.

take up STRIPS. Our goal here is to keep the logical framework, but show that in common cases, projection can be reduced using a form of regression to reasoning about the initial situation, as done in [11]. The reduction is tricky, however, because of the interaction between the various GSFAs and GSSAs, requiring us to solve (auxiliary) projection tasks at each step.

What we propose is a generalized form of regression that is a sensible compromise between syntactic transformations and logical reasoning. Specifically we require the latter only in evaluating the *guards* to decide which GSFAs and GSSAs to apply (see Section 4.1 where regression is again used).

In the following we assume that Σ is an action theory as above, λ is a history, and $\phi(\vec{x}, s)$ and $\psi(\vec{x}, s)$ are sensor-fluent formulas, $\rho(\vec{x}, s)$ is a sensor formula, and $\gamma(\vec{x}, s)$ is a fluent formula. We use the notation $\phi\backslash\lambda$ to mean the formula that results from replacing every sensor function $h_j(s)$ in ϕ by the j-th component of the final sensor reading in λ. We denote by Σ_0 the part of Σ formed by the initial database and the unique name axioms for actions as above.

Lemma 1 *Let $\rho(\vec{x}, s)$ be a sensor formula. Then for every history λ the following statement is valid:*[5]

$$Sensed[\lambda] \supset \forall \vec{x}.\rho(\vec{x}, end[\lambda]) \equiv \rho(\vec{x}, s)\backslash\lambda$$

Proof: By induction on the structure of $\rho(\vec{x}, s)$. ■

To begin, we consider simplifications to formulas resulting from sensing, using the guarded sensed fluent axioms.

Definition 4.1 $\phi(\vec{x}, s)$ simplifies to $\psi(\vec{x}, s)$ at λ *iff there are fluents $F_1(\vec{t_1}, s), \ldots F_k(\vec{t_k}, s)$ in $\phi(\vec{x}, s)$ with $k \geq 0$, and for every $1 \leq i \leq k$, there is a GSFA in Σ*

$$\alpha_i(\vec{z}, s) \supset [F_i(\vec{z}, s) \equiv \rho_i(\vec{z}, s)]$$

where $\Sigma \cup Sensed[\lambda] \models \forall \alpha_i(\vec{t_i}, end[\lambda])$, and $\psi(\vec{x}, s)$ is the result of replacing each $F_i(\vec{t_i}, s)$ in $\phi(\vec{x}, s)$ by $\rho_i(\vec{t_i}, s)\backslash\lambda$.

Definition 4.2 $\phi(\vec{x}, s)$ fully simplifies to $\psi(\vec{x}, s)$ at λ *iff $\phi(\vec{x}, s)$ simplifies to $\psi'(\vec{x}, s)$ at λ and if $\psi'(\vec{x}, s)$ simplifies to $\psi''(\vec{x}, s)$ at λ then $\psi'(\vec{x}, s) = \psi''(\vec{x}, s)$, and $\psi(\vec{x}, s) = \psi'(\vec{x}, s)\backslash\lambda$*

Lemma 2 *If $\phi(\vec{x}, s)$ simplifies to $\psi(\vec{x}, s)$ at λ, then*

$$\Sigma \cup Sensed[\lambda] \models \forall \vec{x}.\phi(\vec{x}, end[\lambda]) \equiv \psi(\vec{x}, end[\lambda])$$

Proof: By logical manipulation and Lemma 1. ■

Next, we consider simplifications involving reasoning backwards using the guarded successor state axioms.

Definition 4.3 $\phi(\vec{x}, s)$ rolls back to $\psi(\vec{x}, s)$ from a non-empty history $\lambda = \lambda' \cdot (A, \vec{\mu})$ *iff $\phi(\vec{x}, s)$ fully simplifies to $\phi'(\vec{x}, s)$ at λ and for every fluent $F_1(\vec{t_1}, s), \ldots F_k(\vec{t_k}, s)$ in $\phi'(\vec{x}, s)$ with $k \geq 0$, there is a GSSA in Σ*

$$\alpha_i(\vec{z}, a, s) \supset [F_i(\vec{z}, do(a, s)) \equiv \gamma_i(\vec{z}, a, s)]$$

where $\Sigma \cup Sensed[\lambda] \models \forall \alpha_i(\vec{t_i}, A, end[\lambda])$, and $\psi(\vec{x}, s)$ is the result of replacing each $F_i(\vec{t_i}, s)$ in $\phi'(\vec{x}, s)$ by $\gamma_i(\vec{t_i}, A, s)$.

[5] We assume a logic with equality.

Lemma 3 *If $\phi(\vec{x}, s)$ rolls back to $\psi(\vec{x}, s)$ from a nonempty history $\lambda = \lambda' \cdot (A, \vec{\mu})$, then*

$$\Sigma \cup Sensed[\lambda] \models \forall \vec{x}.\phi(\vec{x}, end[\lambda]) \equiv \psi(\vec{x}, end[\lambda'])$$

Proof: By logical manipulation and Lemma 2. ■

Putting both forms of simplification together we get:

Definition 4.4 $\phi(\vec{x}, s)$ regresses to $\psi(\vec{x}, s)$ from λ *iff either:*

- *$\lambda = (\vec{\mu_0})$ and $\phi(\vec{x}, s)$ fully simplifies to $\psi(\vec{x}, s)$ at λ.*
- *$\lambda = \lambda' \cdot (A, \vec{\mu})$ and $\phi(\vec{x}, s)$ rolls back to $\psi'(\vec{x}, s)$ from λ, and $\psi'(\vec{x}, s)$ regresses to $\psi(\vec{x}, s)$ from λ'.*

Theorem 4 *If $\phi(\vec{x}, s)$ regresses to $\psi(\vec{x}, s)$ from λ then*

$$\Sigma \cup Sensed[\lambda] \models \forall \vec{x}.\phi(\vec{x}, end[\lambda]) \equiv \psi(\vec{x}, S_0)$$

Proof: By induction on the number of actions in λ using Lemma 2 and Lemma 3. ■

Observe that a formula $\phi(\vec{x}, s)$ can regress to zero, one, or more formulas $\psi(\vec{x}, s)$ from λ, depending on how many entailed guards we can find for the fluents at each stage.

When a formula with a single free variable does regress, then, as a consequence of Theorem 4, we get the following:

Corollary 5 *Under plausible consistency conditions for $\Sigma \cup Sensed[\lambda]$,*[6] *if $\phi(s)$ regresses to $\psi(s)$ from λ then*

$$\Sigma \cup Sensed[\lambda] \models \phi(end[\lambda]) \quad \textit{iff} \quad \Sigma_0 \models \psi(S_0)$$

This provides a *soundness* result for regression: to perform the projection task, it is sufficient to regress the formula, and check the result against the initial database.

Unfortunately, regression in general cannot be *complete*. To see why, suppose nothing is known about fluent F; then a formula like $(F(s) \vee \neg F(s))$ will not regress even though it will be entailed at any history. In Section 5, we show that for certain histories (namely those where enough useful sensing information is available), regression will be complete.

The other drawback of regression as defined is that we need to evaluate guards. However, evaluating a guard is just a sub-projection task, and so for certain "well structured" action theories, we can again apply regression, as we now show.

4.1 Acyclic action theories and g-regression

We now restrict our interest to action theories Σ that are acyclic, in the following sense. Let $\prec$, called *dependency relation*, be a binary relations over fluents s.t. $F' \prec F$ iff there is a GSFA $\{\alpha(\vec{z}, s) \supset [F(\vec{z}, s) \equiv \rho(\vec{z}, s)]\}$ in Σ where F' occurs in $\alpha(\vec{z}, s)$. An action theory Σ *is acyclic* iff the dependency relation $\prec$ is well-founded. When it is, we call the *level* of a fluent F the maximal distance in of $\prec$-chains from a bottom element of $\prec$.

Definition 4.5 *Let Σ be acyclic. The sensor-fluent formula $\phi(\vec{x}, s)$* g-regresses to $\psi(\vec{x}, s)$ from λ *if $\phi(\vec{x}, s)$ regresses to $\psi(\vec{x}, s)$ and*

- *for every simplification step, where a GSFA in Σ*

$$\alpha(\vec{z}, s) \supset [F(\vec{z}, s) \equiv \rho(\vec{z}, s)]$$

is used to replace $F(\vec{t}, s)$ by $\rho(\vec{t}, s)\backslash\lambda'$ for some sub-history λ', we have that the guard $\alpha(\vec{t}, s)$ g-regresses *to some $\alpha'(\vec{t}, s)$ from λ', where $\Sigma_0 \models \forall \alpha'(\vec{t}, S_0)$;*

[6] These will be made explicit in the full paper.

- *for every roll back step, where a GSSA in Σ*
$$\alpha(\vec{z}, a, s) \supset [F(\vec{z}, do(a, s)) \equiv \gamma(\vec{z}, a, s)]$$
is used to replace $F(\vec{t}, s)$ by $\gamma(\vec{t}, A, s)$ for a subhistory $\lambda' \cdot (A, \vec{\mu})$, we have that the guard $\alpha(\vec{t}, A, s)$ g-regresses to some $\alpha'(\vec{t}, A, s)$ from λ', where $\Sigma_0 \models \forall \alpha'(\vec{t}, A, S_0)$.

The trickiest aspect of this definition is to show that the recursion is indeed well-founded. This is done by simultaneous induction on the length of λ and the level of the fluents. Clearly, if a formula g-regresses to another, then it regresses also (although not vice-versa). The main point however is that we only ever need to evaluate formulas at S_0:

Theorem 6 *For an acyclic Σ, under plausible consistency conditions for $\Sigma \cup Sensed[\lambda]$, if $\phi(s)$ g-regresses to some formula from λ, then the only theorem-proving needed to perform projection is to evaluate formulas in Σ_0.*

Proof: By induction on the total number of simplification and roll back steps used to g-regress ϕ, and using Corollary 5. ■

5 JIT-histories

As noted above, we cannot expect to use regression to evaluate sensor-fluent formulas in general: a tautology might be entailed even though nothing is entailed about the component fluents. However, in a practical setting, we can imagine never asking the robot to evaluate a formula unless the history is such that it knows enough about the component fluents, using the given GSSAs and GSFAs, and their component fluents.

For example, assume we have the indoor temperature axioms from Section 3.1. We might only ask the robot to evaluate a formula that mentions the indoor temperature for those histories where the climate control is known to have remained on from some earlier point in the history where the robot was known to be indoors. We do not require the robot to know whether the climate control was on before then (since this may have required going to a control panel), or even whether it is indoors now. In general, we call a history *just-in-time* for a formula, if the actions and sensing readings it contains are enough to guarantee that suitable formulas (including guards) can be evaluated at appropriate points to determine the truth value of all the fluents in the formula. More precisely:

Definition 5.1 *An history λ is a* just-in-time-history *(JIT-history) for a sensor-fluent formula $\phi(\vec{x}, s)$ iff:*

- *$\phi(\vec{x}, s) = \neg\phi_1(\vec{x}, s) \mid \phi_1(\vec{x}, s) \wedge \phi_2(\vec{x}, s)$ and λ is a JIT-history for $\phi_1(\vec{x}, s)$ and $\phi_2(\vec{x}, s)$;*
- *$\phi(\vec{x}, s) = \exists y.\phi_1(y, \vec{x}, s)$ and λ is a JIT-history for the (open) formula $\phi_1(y, \vec{x}, s)$;*
- *$\phi(\vec{x}, s)$ is a sensor formula;*
- *$\phi(\vec{x}, s) = F(\vec{t}, s)$, where F is a fluent, and for some GSFA $\{\alpha(\vec{z}, s) \supset [F(\vec{z}, s) \equiv \rho(\vec{z}, s)]\}$, λ is a JIT-history for $\alpha(\vec{t}, s)$, and $\Sigma \cup Sensed[\lambda] \models \forall \alpha(\vec{t}, end[\lambda])$;*
- *$\phi(\vec{x}, s) = F(\vec{t}, s)$, a fluent, λ is an empty history $(\vec{\mu_0})$, and either $\Sigma_0 \models \forall F(\vec{t}, S_0)$ or $\Sigma_0 \models \forall \neg F(\vec{t}, S_0)$;*
- *$\phi(\vec{x}, s) = F(\vec{t}, s)$, a fluent, $\lambda = \lambda' \cdot (A, \vec{\mu})$, and for some GSSA $\{\alpha(\vec{z}, a, s) \supset [F(\vec{z}, do(a, s)) \equiv \gamma(\vec{z}, a, s)]\}$, λ' is a JIT-history both for $\alpha(\vec{t}, A, s)$ and for $\gamma(\vec{t}, A, s)$, and $\Sigma \cup Sensed[\lambda'] \models \forall \alpha(\vec{t}, A, end[\lambda'])$.*

For JIT-histories we have the following theorem:

Theorem 7 *Let Σ be an action theory as above, and $\phi(\vec{x}, s)$ a sensor-fluent formula. If λ is a JIT-history for $\phi(\vec{x}, s)$, then there exists a formula $\psi(\vec{x}, s)$ such that:*

1. *$\phi(\vec{x}, s)$ regresses to $\psi(\vec{x}, s)$ from λ, and*
2. *if Σ is acyclic, then $\phi(\vec{x}, s)$ g-regresses to $\psi(\vec{x}, s)$ from λ, and either $\Sigma_0 \models \forall \vec{x}.\psi(\vec{x}, S_0)$ or $\Sigma_0 \models \forall \vec{x}.\neg\psi(\vec{x}, S_0)$.*

Proof: (1) is proven by induction on the length of λ and induction on the structure of $\phi(\vec{x}, s)$; (2) is proven by simultaneous induction on the length of λ and the (max) level of the fluents in the $\phi(\vec{x}, s)$, and induction on the structure of $\phi(\vec{x}, s)$. ■

This theorem shows that for JIT-histories, regression is both a sound and complete way of performing projection.

6 An evaluation procedure for projection

Although our action theories are assumed to be *open-world*, a JIT-history provides a sort of *dynamic* closed world assumption in that it ensures that the truth value of any fluent will be known whenever it is part of a formula whose truth value we need to determine. This allows us to evaluate complex formulas as we would if we had a normal closed world assumption. We now consider a Prolog procedure that does this.

We assume the user provides the following clauses:[7]

- `fluent`(F), for each fluent F,
- `sensor`(h), for each sensor function h,
- `ini`(F), for each fluent F such that $\Sigma_0 \models F(S_0)$,
- `closed`(F), for each fluent F such that $\Sigma_0 \models F(S_0)$ or $\Sigma_0 \models \neg F(S_0)$,
- `gsfa`(α, F, ρ), for each GSFA,
- `gssa`(a, α, F, γ), for each GSSA, where a is the distinguished action term quantified in α and γ

Formulas are expressed using `and`(ϕ, ψ), `neg`(ϕ), `equ`(t, t'), and `some`(v, ϕ) where v is a Prolog constant. We drop the situation arguments from fluents and sensor functions in formulas (and keep track of the situation in the history).

Histories are represented as lists. For brevity, we assume a predicate `last`(λ, h, r) which extracts the last value r for sensor function h in history λ. Finally, we assume a predicate `sub`(v, x, ϕ, ϕ') with the meaning that ϕ' is the formula obtained by substituting x for v in the formula ϕ (see [6]).

Now we define `eval`(λ, ϕ, b), with the intended meaning that the formula ϕ evaluates to the truth-value b (`tt`/`ff`) for the history λ, as follows:

```
eval(H,and(P1,P2),tt) :-
   eval(H,P1,tt), eval(H,P2,tt).
eval(H,and(P1,P2),ff) :-
   eval(H,P1,ff); eval(H,P2,ff).
```

[7]For simplicity in what follows, we do not distinguish between situation calculus formulas and their representations as Prolog terms.

```
eval(H,neg(P),tt) :- eval(H,P,ff).
eval(H,neg(P),ff) :- eval(H,P,tt).

eval(H,some(V,Pv),tt) :-
   sub(V,_,Pv,Px), eval(H,Px,tt).
eval(H,some(V,Pv),ff) :-
   not(sub(V,_,Pv,Px), not eval(H,Px,ff)).
   /* double negation for ''for all'' */
   /* so not eval(H,Px,ff) flounders!  */

eval(H,equ(E,V),tt) :-
   sensor(E), last(H,E,R), R=V.
eval(H,equ(E,V),ff) :- /* Neg as failure */
   sensor(E), last(H,E,R), not R=V.

eval([(Mu)],F,tt) :- fluent(F), ini(F).
eval([(Mu)],F,ff) :- /* Neg as failure */
   fluent(F), closed(F), not ini(F).

eval(H,F,B) :-
   fluent(F), gsfa(Alpha,F,Rho),
   eval(H,Alpha,tt), eval(H,Rho,B).

eval([(A,Mu)|H],F,B) :-
   fluent(F), gssa(A,Alpha,F,Gamma),
   eval(H,Alpha,tt), eval(H,Gamma,B).
```

Observe that a formula $\mathtt{eval}(\lambda, \phi, b)$ can either succeed returning `tt`, succeed returning `ff`, fail, or not terminate. Under the assumption that all the auxiliary predicates are correct and terminating, we get the following soundness and a weak form of completeness for `eval`:

Theorem 8 *Assume that* $\mathtt{eval}(\lambda, \phi(\vec{x}, s), \mathtt{B})$ *succeeds with computed answer* $\vec{x}{=}\vec{t}$, $\mathtt{B}{=}b$. *If* $b{=}\mathtt{tt}$, *then* $\Sigma \cup Sensed[\lambda] \models \forall\phi(\vec{t}, end[\lambda])$*; if* $b{=}\mathtt{ff}$ *then* $\Sigma \cup Sensed[\lambda] \models \forall\neg\phi(\vec{t}, end[\lambda])$.

Proof: By induction on the structure of $\phi(\vec{x}, s)$. ■

Theorem 9 *Let* λ *be a JIT-history for* $\phi(\vec{x}, s)$. *Then,* $\mathtt{eval}(\lambda, \phi(\vec{x}, s), \mathtt{B})$ *does not finitely fail.*

Proof: By induction on the structure of $\phi(\vec{x}, s)$. ■

Note that we cannot guarantee termination since we can get into a loop evaluating guards of GSFAs or GSSAs, or by floundering in trying to evaluate to `ff` an existential. We can eliminate the first problem by using acyclic action theories. For the second, we can close the domain. Let $\mathtt{domain}(o)$ be a user-defined predicate over a finite domain. We can then change the definition of `eval` for existentials as follows:

```
eval(H,some(V,Pv),tt) :-
   domain(O), sub(V,O,Pv,Po), eval(H,Po,tt).
eval(H,some(V,Pv),ff) :-
   not(domain(O), sub(V,O,Pv,Po),
      not eval(H,Po,ff)).
```

For this new version of `eval` we get a completeness result:

Theorem 10 *Let* Σ *be acyclic, let* $\phi(s)$ *be a sensor-fluent formula with no free variables except the situation argument* s, *and let* λ *be a JIT-history for* $\phi(s)$. *Then,* $\mathtt{eval}(\lambda, \phi(s), \mathtt{B})$ *always succeeds, returning either* `tt` *or* `ff`.

So under these circumstances, `eval` is a sound and complete implementation of projection.

7 Conclusions

We have given a formal definition of progression for a generalized action theory where successor state axioms and sensing information are only conditionally applicable. We also showed that in certain circumstances, a regression-based evaluation procedure could correctly perform the task.

Many open problems remain, however. How can we decide in an automated but practical way when regression can be used? It may be expecting too much of a conditional planner to determine what actions it should perform now to permit it to later use sensing information in this way. An interesting alternative is offered by the high-level program execution model. Given a program like "if ϕ then do δ_1 else do δ_2," the *user* can take the responsibility of inserting a prior program ensuring that the resulting history is just in time for ϕ.

Another related problem is the projection of initial databases. Once a robot actually performs a sequence of actions in the world, we would prefer to no longer regress all the way back to the initial situation, but instead to project the database forward to the current state [8]. How this can be done for the action theories we are proposing remains to be seen.

References

[1] C. Baral and T.C. Son. Approximate reasoning about actions in presence of sensing and incomplete information. In *Proc. of ILPS'97*, 387–401.

[2] R. Fikes and N. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. In *Artificial Intelligence*, 2, 189–208, 1971.

[3] K. Golden and D.S. Weld. Representing sensing actions: the middle ground revisited. In *Proc. of KR'96*, 174–185.

[4] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, vol. 4, Edinburgh University Press, 1969.

[5] H. Levesque. What is planning in the presence of sensing? In *Proc. of AAAI'96*, 1139–1146.

[6] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl. GOLOG: A logic programming language for dynamic domains. In *Journal of Logic Programming*, 31, 59–84, 1997.

[7] F. Lin and R. Reiter. State constraints revisited. *Journal of Logic and Computation*, 4(5), 655–678, 1994.

[8] F. Lin and R. Reiter. How to progress a database. *Artificial Intelligence*, 92, 131–167, 1997.

[9] D. Poole. Logic programming for robot control. In *Proc. IJCAI'95*, 150–157.

[10] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In *Artificial Intelligence and Mathematical Theory of Computation*, 359–380. Academic Press, 1991.

[11] R. Reiter. *Knowledge in Action: Logical Foundation for Describing and Implementing Dynamical Systems*. In preparation.

[12] D.S. Weld, C.R. Anderson, D.E. Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proc. of AAAI'98*, 897–904.