Recovering from Modeling Faults in GOLOG

Marcus Bjäreland
Dept of Comp and Info Science
Linköping University
S-581 83 Linköping, Sweden
marbj@ida.liu.se

Abstract

In this paper we study the problem of detection, classification and recovery from discrepancies due to faulty modeling of controlled systems. We do this by extending the existing Execution Monitoring mechanism for the high-level robot programming language GOLOG [De Giacomo *et al.*, 1998] by introducing a revision technique that transforms the faulty underlying situation calculus theory to a theory that reflects the controlled system more adequately.

1 Introduction

Execution Monitoring (EM) has been widely studied by AI researchers (see e.g. [Ambros-Ingerson and Steel, 1988; Knoblock, 1995; Simmons *et al.*, 1997; Doyle *et al.*, 1986; Beetz and McDermott, 1994]) and can be defined as an agent's process of identifying discrepancies between observations of the actual world and the prediction and expectations derived from its representation of the world, classifying such discrepancies, and recover from them. While most of the previous work on EM has focused on monitoring of plan execution, and recovery by repairing failed plans, we are interested in the detection, classification and recovery from discrepancies due to *faulty expectations*. We will use ideas from Ontological Control [Bjäreland and Fodor, 1998; Fodor, 1998], where the fundamental problem can be stated as "Given a detected discrepancy, when is it possible to correctly classify the discrepancy as being due to a faulty model (faulty expectation)?". This question has also, though to a smaller degree, interested researchers concerned with machine learning of planning operators (e.g. [Benson, 1996; Wang, 1994]). Their work has focused on how to improve the models with learning techniques from discrepancies (negative instances) and positive instances provided by a teacher. The classification in theses approaches are typically performed when the same operator has failed repeatedly, e.g. in Benson's TRAIL system [1996] an operator has to fail four times before the learning mechanism is invoked. Unfortunately, many domains are unforgiving and repeated failures of an action may have disastrous consequences. The revision technique we propose is quite different from machine learning techniques since we only consider negative instances, and we do not generalize from multiple discrepancies.

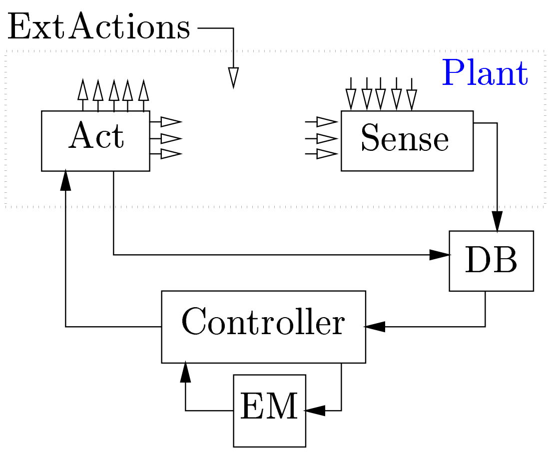

Figure 1: The control system

In Figure 1 a generic control system is depicted. The controller sends signals to the actuators, which perform some action on the plant. The actuators are equipped with sensors whose values are sent to a database together with the sensor values from the plant. The database is assumed to filter the sensor values and monitor the ranges of the sensor values. The controller gets its input vector from the database, and reacts to the input by sending a signal to the actuators. We also assume that there are external actions (or, disturbances) that may change features of the plant in unpredictable ways. The box labeled EM represents the execution monitoring mechanism and is the concern of this paper.

In Ontological Control perfect actuators are assumed in thye following sense: If an actuator performs an action, the expected results, or values, measured by the actuator sensors are assumed to materialize. This, together with the assumption that actions are "energized" (that an action is executed exactly as long as its precondition is true), means that if we detect a discrepancy there can be two cases: either the actuators performed perfectly, but failed to move the system into the expected state, or they did not perform at all, i.e. they did not perform in such a way that the part of the precondition constraining the values of the actuator sensors turned false, but the system changed state anyhow. In the first case, we say that the discrepancy is due to a *violation of ontological assumptions* (VOA), which is motivated by our trust in the actuators. In the second case the discrepancy is said to to be caused by an *external action* (EA), since we assume that control actions (i.e. actions invoked by the controller) are the only actions that can change the actuator-sensor values.

When EA occurs we have no reason to believe that there is something wrong with the model. Thus, we can use the model to recover, for example by doing replanning. However, VOAs are a bit more serious, and some type of revision of the model is necessary to make it more adequately reflect the controlled system.

In this paper we formalize ontological control in situation calculus, and show how ontological control can enhance the previously proposed execution monitoring mechanism in Golog [De Giacomo *et al.*, 1998].

2 Situation Calculus

The situation calculus (SitCalc) is a sorted second-order language with equality with four sorts: primitive actions, situations, fluents, and objects. A fluent is a term whose value might vary in different situations. We use $H(F(\vec{o}), s)$ to state that the fluent $F(\vec{o})$ is true in situation s, for a vector, $\vec{o}$, of object constants. Sometimes we write F and A to denote $F(\vec{x})$ and $A(\vec{x})$.

The set of all primitive actions for a particular theory will be denoted $\mathcal{A} = \{A_1, \ldots, A_n\}$, the set of fluents $\mathcal{F} = \{F_1, \ldots, F_m\}$, where every fluent has a particular arity, and the set of objects $\mathcal{O} = \{o_1, \ldots, o_k\}$. The set of fluents whose values depend only on the actuator sensors is denoted $Conf$. The set of background axioms of SitCalc is denoted Σ and we refer interested readers to [Reiter, 1993] for further details about the logic. For a particular axiomatization, the following axioms must be provided:

1. **Action precondition axioms**, one for each primitive action $A(\vec{x})$, having the syntactic form
$$Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s).$$
2. **Successor state axioms**, one for each fluent F, having the syntactic form:
$$Poss(a, s) \supset (H(F(\vec{x}), do(a, s)) \equiv \gamma_F^+(\vec{x}, a, s) \vee (H(F(\vec{x}), s) \wedge \neg\gamma_F^-(\vec{x}, a, s))),$$
where $\gamma_F^+(\vec{x}, a, s)$, and $\gamma_F^-(\vec{x}, a, s)$ are formulae with free variables among $\vec{x}$, a, s, and whose only situation term is s. Moreover, it is assumed that $\gamma_F^+(\vec{x}, a, s)$ and $\gamma_F^-(\vec{x}, a, s)$ never are simultaneously true in any model of Σ. The formulae γ_F^+ and γ_F^- denote the conditions under which the fluent F changes to true and false respectively, in situation $do(a, s)$.
For simplicity of the presentation, we also assume that γ_F^+ and γ_F^- are on the form
$$\bigvee_i \pi_i(\vec{x}, s) \wedge a = A_i(\vec{x}). \quad (1)$$
Intuitively, $\pi_i(\vec{x}, s)$ denotes the condition under which action A_i changes the value of F.
Without loss of generality, we also assume that there exists exactly one disjunct in the γs for every action A_i.
3. **Initial situation axioms** – what is true initially, before any actions have occurred. This is any finite set of sentences that mention only the situation term S_0, or that are situation independent.

Thus, an application axiomatization is a set $\Delta = \Sigma \cup \Gamma_{AP} \cup \Gamma_{SSA} \cup \Gamma_{S_0}$, where Σ is the set of background axioms, Γ_{AP} is a set of action precondition axioms, Γ_{SSA} is a set of successor state axioms, and Γ_{S_0} is a set of initial situation axioms and all situation independent axioms.

We define an *observation in situation* s, O_s, to be the set of all *ground* fluents that are true in situation s. It is necessary that observations are finite, which is imposed by not allowing other function symbols than *do*. For an observation $O_s = \{F_1, \ldots, F_n\}$ we write $\bigwedge O_s$ to denote the formula $\bigwedge_{i=1}^n H(F_i, s)$. By $\bigwedge O_{s/s'}$ we denote $\bigwedge_{i=1}^n H(F_i, s')$.

To be able to use the techniques described below we need to ensure that the controller is not *logging*, i.e. that the successful execution of an action always changes the values of some fluents in $Conf$. Formally, we assume that
$$\begin{array}{l} \{F(\vec{o}) \mid F \in Conf \text{ and } F(\vec{o}) \in O_s\} \\ \neq \\ \{F(\vec{o}) \mid F \in Conf \text{ and } F(\vec{o}) \in O_{do(A,s)}\}, \end{array} \quad (2)$$
for any action A and situation s.

3 Golog

Golog is a high-level logic programming language used for agent-based programming. Detailed discussion of Golog can be found in [De Giacomo *et al.*, 1998], so we will only describe some principles behind the Golog language and the interpretation process. For Golog programs δ_1, δ_2, we have the following examples of constructs:

$(\delta_1; \delta_2)$,	sequential execution
$(\delta_1 \mid \delta_2)$,	nondeterministic choice
δ_1^*,	nondeterministic iteration

The semantics of Golog is a transition semantics, where the predicate $Trans(\delta, s, \delta', s')$, given a program δ and a situation s, tells us which is a possible next step in the computation, returning the resulting situation s' and the program δ' that remains to executed.

The formal definitions of $Trans$ can be found in [De Giacomo *et al.*, 1998].

Execution Monitoring is handled by the predicate $Monitor(\delta, s, \delta', s')$, which checks for the existence of an exogenous program, determines the situation s' reached by this program, and if the monitored program δ terminates offline, the monitor returns δ, else it invokes a recovery mechanism that determines a new program δ'. The suggested recovery mechanism tries to find a sequence of primitive actions p that would move the system into a situation where the original program δ can be successfully executed, i.e. it generates a program $p; \delta$ if such a program exists.

Discussion

In the current implementation of execution monitoring (EM) for Golog, the EM senses occurrences of "exogenous" actions. In fact the actions themselves are not exogenous, it is the *occurrences* that are, i.e. the EM senses the occurrence of a Golog program which is not the program Golog is currently executing. As argued in [De Giacomo *et al.*, 1998], this "seems dubious in practice", since it is more likely that an EM would detect *effects* of actions, rather than the occurrences. In the paper, the authors claim to reconcile this point by proposing to introduce new, fictitious actions, one for each fluent, whose effects are to alter their corresponding fluents' truth value.

We claim that this is still a problematic issue and that the proposed solution is unconvincing, since it considerably weakens the axiomatization. Every fluent value change can be explained by a fictitious action occurrence, so it is no longer possible to claim that an axiomatization fails to model an application adequately. Also, the proposal limits the possibility of using diagnosis techniques along the lines of those suggested in [McIlraith, 1998], since the number of possible action sequences that could explain a discrepancy is exponentially larger than in the original axiomatization.

4 Our Proposal

We suggest that the EM should sense fluent values instead of action occurrences. The $Trans$ function makes it possible to compute the expected result of a primitive action, so it is easy to determine whether there is a discrepancy between the sensed values and the expected ones. Next, if a discrepancy is detected we know that the observation is not a consequence of the axiomatization, since the initial state is completely specified and the actions are deterministic. To proceed, we suggest that the *model* (the underlying SitCalc axiomatization) should be changed in case of a discrepancy. We can classify the possible causes of the discrepancy as either being due to VOA, which implies that the successor state axioms do not reflect the environment adequately, or due to EA, which implies that the successor state axioms still are corroborated, but that the system is in a different state than what was expected. The changes of the model that we propose are:

Type	O_{prev}	O_{exp}	O_{mat}	Revision Action
A	F	F	$\neg F$	Add more information to γ_F^-
B	$\neg F$	$\neg F$	F	Add more information to γ_F^+
C	F	$\neg F$	F	Remove information from γ_F^-
D	$\neg F$	F	$\neg F$	Remove information from γ_F^+

Table 1: The four kinds of discrepancies.

For EA: Replace the initial situation axioms with the new and unexpected observation, where the situation argument of the fluents is relativized to situation S_0, and restart the execution. Invoke the recovery mechanism if necessary.

For VOA: Let O_{prev} be the state of the previous situation, A the primitive action invoked in the previous situation, O_{exp} the expected state after invoking A in a situation with state O_{prev}, and O_{mat} the observed current state. For a particular fluent F with successor state axiom we have four types of discrepancies described in Table 1. Types A and B describe the cases where the fluent F is modeled to be inert (i.e. to not change its value) by the action A, but that the value is sensed to have changed. This means that, in the case of a VOA, we need to add the particular case to γ_F^+ or γ_F^-, to make sure that the observed change is considered by the model. For cases C and D, changes are modeled but do not occur. This means that the changes assumed by the model should be removed. When the changes have been performed, we replace all initial situation axioms with the materialized observation and continue (possibly with recovery).

5 Formalization of the Idea

Assume that for an application axiomatization $\Delta = \Sigma \cup \Gamma_{AP} \cup \Gamma_{SSA} \cup \Gamma_{S_0}$ we have an observation $O_{\mathbf{s}}$ and that we perform the action A in $\mathbf{s}$ to take us to situation $do(A, \mathbf{s})$, and that we expect to sense the observation $O_{do(A,\mathbf{s})}$, but that we instead sense an observation $O'_{do(A,\mathbf{s})}$, where $O_{do(A,\mathbf{s})} \neq O'_{do(A,\mathbf{s})}$ (i.e. we have detected a discrepancy).

There are two possible causes for the discrepancy, EA or VOA.

EA

Now, if

$$\{\, F(\vec{o}) \mid F \in Conf \text{ and } F(\vec{o}) \in O_{\mathbf{s}} \,\} = \{\, F(\vec{o}) \mid F \in Conf \text{ and } F(\vec{o}) \in O'_{do(A,\mathbf{s})} \},\quad (3)$$

we know that the actuator sensors have not changed from situation $\mathbf{s}$ to $do(A, \mathbf{s})$, and we explain the the discrepancy with EA. We then change the application axiomatization to $\Delta' = (\Delta - \Gamma_{S_0}) \cup \{\bigwedge O'_{do(A,\mathbf{s})}/S_0\}$. That is, we replace the initial situation axioms by the latest observation.

VOA

We detect a VOA by noticing that

$$\{\, F(\vec{o}) \mid F \in Conf \text{ and } F(\vec{o}) \in O_{do(A,\mathbf{s})} \,\} \\ = \\ \{\, F(\vec{o}) \mid F \in Conf \text{ and } F(\vec{o}) \in O'_{do(A,\mathbf{s})} \,\}, \quad (4)$$

i.e. that the actuator sensors have changed exactly as expected, but other changes have not followed our expectations.

Given the three observations $O_{\mathbf{s}}$, $O_{do(A,\mathbf{s})}$, and $O'_{do(A,\mathbf{s})}$, we can easily classify every discrepancy as being of type A, B, C, or D.

Let $F(\vec{o})$ be a discrepancy in situation $do(A,s)$, and $H(F(\vec{x}), do(a,s)) \equiv \ldots$ the corresponding successor state axiom, where the length of $\vec{x}$ is n. To construct the formula needed for model repair, as in the fifth column in table 1, we need the formula describing the sensed observation in situation $\mathbf{s}$ (the *precondition* of the detected VOA), i.e. $\bigwedge O_{\mathbf{s}}$, and the *instantiation formula*, $inst^{\vec{o}}_{\vec{x}}$ of the variables in $\vec{x}$, which is $inst^{\vec{o}}_{\vec{x}} \equiv x_1 = o_1 \wedge \ldots \wedge x_n = o_n$. The instantiation formula describes the variable bindings materialized in situation $do(A,s)$. For each discrepancy $F(\vec{o})$ in situation $\mathbf{s}$ we construct the formula $\beta^{\mathbf{s}}_{F(\vec{o})} \equiv \bigwedge O_{\mathbf{s}/s} \wedge inst^{\vec{o}}_{\vec{x}}$.

We assume that the successor state axiom for F is on the form

$$\begin{aligned} H(F(\vec{x}), do(a,s)) \equiv& \\ &\bigvee_i \pi^+_i(\vec{x},s) \wedge a = A_i &(5)\\ &\vee (H(F(\vec{x}),s) \wedge \\ &\neg \bigvee_i \pi^-_i(\vec{x},s) \wedge a = A_i). &(6) \end{aligned}$$

We formalize the repair actions mentioned in the fifth column in table 1 as follows:

Type A: Replace the disjunct $\pi^-(\vec{x},s) \wedge a = A$ in (6) by

$$(\pi^-(\vec{x},s) \vee \beta^s_{F(\vec{o})}) \wedge a = A.$$

Type B: Replace the disjunct $\pi^+(\vec{x},s) \wedge a = A$ in (5) by

$$(\pi^+(\vec{x},s) \vee \beta^s_{F(\vec{o})}) \wedge a = A.$$

Type C: Replace the disjunct $\pi^-(\vec{x},s) \wedge a = A$ in (6) by

$$\pi^-(\vec{x},s) \wedge \neg\beta^s_{F(\vec{o})} \wedge a = A.$$

Type D: Replace the disjunct $\pi^+(\vec{x},s) \wedge a = A$ in (5) by

$$\pi^+(\vec{x},s) \wedge \neg\beta^s_{F(\vec{o})} \wedge a = A.$$

Observe that the structure of (6) and (5) remains unchanged after the the repair actions. To make sure that the application order of the repair actions does not matter, we need the following lemma.

Lemma 1 For two discrepancies, $F(\vec{o})$ of type A and $F(\vec{o'})$ of type C, the order in which we apply the revision actions is irrelevant.

Proof: We can see that the repair actions give syntactically different resulting formulae depending on the order of the repairs, i.e. if we use the type A repair before type C, the result will be

$$(\pi^-(\vec{x},s) \vee \beta^s_{F(\vec{o})}) \wedge \neg\beta^s_{F(\vec{o'})} \wedge a = A,$$

and if we start with type C repair we get

$$((\pi^-(\vec{x},s) \wedge \neg\beta_{F(\vec{o'})}) \vee \beta^s_{F(\vec{o})}) \wedge a = A.$$

The only case when their respective truth values differ is when $\beta_{F(\vec{b})}$ and $\beta_{F(\vec{c})}$ are both true, which would imply that $inst^{\vec{o}}_{\vec{x}} \wedge inst^{\vec{o'}}_{\vec{x}}$ is satisfiable. This can only occur if $\vec{o} = \vec{o'}$, which means that both $F(\vec{o})$ and $\neg F(\vec{o})$ have materialized (since we had one type A and one type C discrepancy). This is obviously false, so the two formulae are equivalent.□

For discrepancies of type B and D, the proof is analogous.

To summarize, we detect a discrepancy, classify it as caused by VOA, then determine the discrepancy type of all discrepancies. Based on the type information we perform revision on the successor state axioms of the discrepancies. When all discrepancies have been dealt with, we have a new set of successor state axioms Γ'_{SSA}. We replace the previous set Γ_{SSA} in Δ with Γ'_{SSA} and replace the initial situation axioms in Δ with the materialized observation, i.e. the new application axiomatization is defined as

$$\Delta' = ((\Delta - \Gamma_{S_0}) \cup O'^{S_0}_{do(A,\mathbf{s})}) \cup ((\Delta - \Gamma_{SSA}) \cup \Gamma'_{SSA}).$$

Note that the distinction between EA and VOA strongly depends on the non-logging assumption formalized in Equation 2. If the configuration parts of two consecutive observations would not differ, the classification would not work.

6 Trolley Example

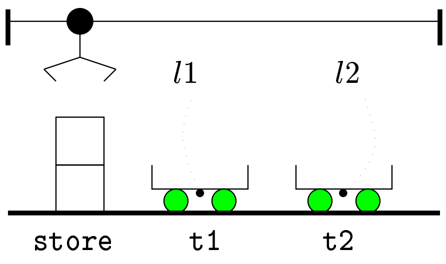

Figure 2: The plant.

We will now look at an example of a plant depicted in Figure 2. We control a robot arm (our actuator) that is supposed to move boxes from `store` to the trolleys at positions t1 and t2. The actuator output is the value of a fluent $Position(p, s)$ (for the position of the robot arm) p can take the values store, t1, or t2, representing the depicted possible positions. The plant signals give values to a fluent $Load(l, n)$, where l denotes the sensor ($l1$ or $l2$), and n denotes the number of boxes on the respective trolley, i.e. 0, 1, or 2. We have one control action, $Move(x, y)$, where x and y are two different positions, and we assume that when x is store, y is t1 or t2, and there are less than 2 boxes on the trolleys, $Move(x, y)$ invokes the robot arm to grab one box at store and move to t1 or t2 and to drop the box at that position. We choose the following SitCalc axiomatization:

$$\begin{aligned}
&Poss(Move(x, y), s) \equiv \\
&\quad x \neq y \wedge (y = \mathtt{store} \vee (x = \mathtt{store} \wedge \\
&\quad \forall n, n'. (H(Load(l1, n), s) \wedge \\
&\quad H(Load(l2, n'), s) \rightarrow n + n' < 2))),
\end{aligned}$$

We define the successor state axiom for our only member of $Conf$, $Position$, as

$$\begin{aligned}
&H(Position(p), do(a, s)) \equiv \\
&\quad \exists x. a = Move(x, p) \vee \\
&\quad (H(Position(p), s) \wedge \neg\, a = Move(p, x)).
\end{aligned}$$

The successor state axiom for $Load$ is somewhat larger:

$$\begin{aligned}
&H(Load(l, n), do(a, s)) \equiv \\
&\quad (\quad (l = l1 \wedge n = 0 \wedge \mathbf{F}) \vee \\
&\quad (\quad ((l = l1 \wedge n = 1 \wedge H(Load(l, 0), s)) \vee \\
&\qquad (l = l1 \wedge n = 2 \wedge H(Load(l, 1), s))) \wedge \\
&\qquad a = Move(\mathtt{store}, \mathtt{t1}) \quad) \vee \\
&\quad (l = l2 \wedge n = 0 \wedge \mathbf{F}) \vee \\
&\quad (\quad ((l = l2 \wedge n = 1 \wedge H(Load(l, 0), s)) \vee \\
&\qquad (l = l2 \wedge n = 2 \wedge H(Load(l, 2), s))) \wedge \\
&\qquad a = Move(\mathtt{store}, \mathtt{t2}) \quad) \\
&\quad) \quad \vee \\
&\quad (H(Load(l, n), s) \wedge \\
&\quad \neg\, ((l = l1 \wedge a = Move(\mathtt{store}, \mathtt{t1})) \vee \\
&\qquad (l = l2 \wedge a = Move(\mathtt{store}, \mathtt{t2})))).
\end{aligned}$$

Notice that the axiom closely follows the schema of Equation (1).

Initially, the arm is at position store and there are no boxes on the trolleys, i.e.

$$\begin{aligned}
&\Gamma_{S_0} = \\
&\quad \{H(Position(\mathtt{store}), S_0), H(Load(l1, 0), S_0), \\
&\quad H(Load(l2, 0), S_0)\}
\end{aligned}$$

Now, assume that we execute $Move(\mathtt{store}, \mathtt{t1})$ in S_0. Let $\mathbf{s} = do(Move(\mathtt{store}, \mathtt{t1}), S_0)$. We then expect the observation

$$O_{\mathbf{s}} = \{Position(\mathtt{t1}), Load(l1, 1), Load(l2, 0)\}.$$

to be sensed. We will now illustrate the two causes of discrepancies:

EA

Assume that somebody moves a box from the store to the trolley with load sensor $l1$ when the robot arm has begun executing $Move(\mathtt{store}, \mathtt{t1})$. Since actions are energized, and the exogenous move will make the precondition of the action false (i.e. the state in which the action was invoked), the action will stop, and we will sense

$$O'_{\mathbf{s}} = \{Position(\mathtt{store}), Load(l1, 1), Load(l2, 0)\}.$$

We detect the discrepancy by noting that $O_{\mathbf{s}} \neq O'_{\mathbf{s}}$, and classify the cause of the discrepancy as EA with Equation (3). To continue, we replace Γ_{S_0} by

$$\begin{aligned}
&\{\bigwedge O'_{\mathbf{s}/S_0}\} = \\
&\quad \{H(Position(\mathtt{store}), S_0) \wedge H(Load(l1, 1), S_0) \wedge \\
&\quad H(Load(l2, 0), S_0)\}
\end{aligned}$$

and try to recover the Golog program.

VOA

An unmodeled ontological assumption of the system is that position t1 corresponds to load sensor $l1$, and that t2 corresponds to $l2$. If this is false the model is not valid for control. Thus, we assume that the position of the trolleys has been changed, which means that the sensed observation after performing $Move(\mathtt{store}, \mathtt{t1})$ in S_0 will be

$$O'_{\mathbf{s}} = \{Position(\mathtt{t1}), Load(l1, 0), Load(l2, 1)\}$$

Again, we detect the discrepancy by noticing that $O_{\mathbf{s}} \neq O'_{\mathbf{s}}$, and classify at as caused by VOA with Equation (4). We have four discrepancies, where $Load(l1, 1)$ is of type D, $Load(l2, 0)$ is of type A, $Load(l1, 0)$ is of type B, and $Load(l2, 1)$ is of type C. The precondition of the VOA is in this case

$$\begin{aligned}
&\bigwedge O_{S_0/s} \equiv \\
&\quad H(Position(\mathtt{store}), s) \wedge H(Load(l1, 0), s) \wedge \\
&\quad H(Load(l2, 0), s).
\end{aligned}$$

Thus, we have

$$\beta^{S_0}_{Load(l1,1)} \equiv \bigwedge O_{S_0/s} \wedge \underbrace{l = l1 \wedge n = 1}_{inst^{l1,1}_{l,n}}.$$

Similarly, we can construct the β^s_F for each discrepancy F.

When we repair the successor state axiom for $Load$ we get the following result:

$$
\begin{aligned}
&H(Load(l,n), do(a,s)) \equiv \\
&\quad (\\
&\quad (l = l1 \wedge n = 0 \wedge \mathbf{F}) \vee \\
&\quad (\quad ((l = l1 \wedge n = 1 \wedge H(Load(l,0), s)) \vee \\
&\qquad (l = l1 \wedge n = 2 \wedge H(Load(l,1), s)) \vee \beta^{S_0}_{Load(l1,0)}) \wedge \\
&\qquad \neg\beta^{S_0}_{Load(l1,1)} \wedge a = Move(\mathtt{store}, \mathtt{t1}) \quad) \vee \\
&\qquad \ldots \\
&\quad (H(Load(l,n), s) \wedge \\
&\quad \neg (((l = l1 \vee \beta^{S_0}_{Load(l2,0)}) \wedge \neg\beta^{S_0}_{Load(l2,1)} \wedge \\
&\qquad a = Move(\mathtt{store}, \mathtt{t1})) \vee \\
&\qquad (l = l2 \wedge a = Move(\mathtt{store}, \mathtt{t2})))).
\end{aligned}
$$

It is easy to see that this particular VOA cannot occur again. It will be handle properly by the new successor state axiom. However, if we perform the action $Move(\mathtt{store}, \mathtt{t2})$ a similar VOA will occur.

7 Discussion

There are basically three assumptions that are necessary for adapting ontological control to GOLOG. We will discuss them one at a time:

Trust in the Actuators

We assume that the expected values of the fluents corresponding to actuator sensors always materializes iff the actions are completely executed. For typical AI research platforms, such as mobile robots, this is an unrealistic assumption, since neither the sensors, nor the actuators are robust enough. Also, the typical AI environments are very complex and unpredictable. *However, in industrial application, e.g. in process control, the assumption is not only realistic, it is also necessary for the applications to be usable.* For example, in ABB Industrial Product's cold mill flatness controller, STRESSOMETER, the flatness over the profile of a sheet of steel cannot differ more that 50 μm.

Actions are Energized

We assume that an action is executed exactly as long as its precondition is true. An alternative to energized actions are "ballistic" actions, where the action is invoked and "left on its own" until it is stops by itself or is stopped by another action. For controllers that make their decisions based on the continuous stream of sensor readings, rather than on asynchronously appearing events or messages, it is more natural to use the energized action view.

No Logging

Two consecutive states are assumed to have mutually exclusive configurations. This means that the controller is not allowed to simply track the execution of actions, without changing the actions. However, a possible solution to this problem would be to collapse logging states into one. Then some information is lost, but not necessarily any information relevant for EM.

The assumptions discussed above are not necessary for the EM mechanism presented in [De Giacomo *et al.*, 1998]. The reason for imposing them is to be able to use ideas from Ontological Control. However, we believe that the techniques introduced in this paper can be used for model revision independently of Ontological Control.

8 Conclusion

In this paper we have studied the problem of detection, classification and recovery from discrepancies due to faulty modeling of controlled systems. By adopting Ontological Control to the high-level robot programming language GOLOG, we have enhanced the previously existing approach to EM for GOLOG, and introduced a model revision technique for discrepancies due to faulty underlying SitCalc theories.

References

[Ambros-Ingerson and Steel, 1988] J. Ambros-Ingerson and S. Steel. Intergrating planning, execution and monitoring. In *AAAI'88*.

[Beetz and McDermott, 1994] M. Beetz and D. McDermott. Improving robot plans during their execution. In *AIPS94*.

[Benson, 1996] S. Benson. *Learning Action Models for Reactive Autonomous Agents.* Phd thesis, Department of Computer Science, Stanford University, 1996.

[Bjäreland and Fodor, 1998] M. Bjäreland and G. Fodor. Ontological control. In *DX'98*.

[De Giacomo *et al.*, 1998] G. De Giacomo, R. Reiter, and M. Soutchanski. Execution monitoring of high-level robot programs. In *KR'98*.

[Doyle *et al.*, 1986] R. Doyle, D. Atkinson, and R. Doshi. Generating perception requests and expectations to verify the execution of plans. In *AAAI'86*.

[Fodor, 1998] G. Fodor. *Ontologically Controlled Autonomous Systems: Principles, Operations and Architecture.* Kluwer Academic, 1998.

[Knoblock, 1995] C. Knoblock. Planning, executing, sensing, and replanning for information gathering. In *IJCAI'95*.

[McIlraith, 1998] S. McIlraith. Explanatory diagnosis: Conjecturing actions to explain observations. In *KR'98*.

[Reiter, 1993] R. Reiter. Proving properties of states in the situation calculus. *Artificial Intelligence*, 64:337–351, December 1993.

[Simmons *et al.*, 1997] R. Simmons, R. Goodwin, K. Haigh, S. Koenig, J. O'Sullivan, and M. Veloso. XAVIER: Experience with a layered robot architecture. *SIGART Bulletin*, 8(1-4):22–33, 1997.

[Wang, 1994] X. Wang. Learning planning operators by observation and practice. In *AIPS94*.