Belief Management for High-Level Robot Programs

Stephan Gspandl, Ingo Pill, Michael Reip, Gerald Steinbauer
Institute for Software Technology
Graz University of Technology
Graz, Austria
{sgspandl,ipill,mreip,steinbauer}@ist.tugraz.at

Alexander Ferrein[*]
Knowledge-Based Systems Group
RWTH Aachen University
Aachen, Germany
ferrein@cs.rwth-aachen.de

Abstract

The robot programming and plan language IndiGolog allows for on-line execution of actions and offline projections of programs in dynamic and partly unknown environments. Basic assumptions are that the outcomes of primitive and sensing actions are correctly modeled, and that the agent is informed about all exogenous events beyond its control. In real-world applications, however, such assumptions do not hold. In fact, an action's outcome is error-prone and sensing results are noisy. In this paper, we present a belief management system in IndiGolog that is able to detect inconsistencies between a robot's modeled belief and what happened in reality. The system furthermore derives explanations and maintains a consistent belief. Our main contributions are (1) a belief management system following a history-based diagnosis approach that allows an agent to actively cope with faulty actions and the occurrence of exogenous events; and (2) an implementation in IndiGolog and experimental results from a delivery domain.

1 Introduction

High-level control is a serious challenge in autonomous systems design, especially for complex and dynamic domains. In the cognitive robotics domain, the framework around the logic-based robot programming and plan language IndiGolog [Giacomo *et al.*, 2009] (an on-line variant of the Golog family [Levesque *et al.*, 1997]) provides the means for state-of-the-art high-level control. However, basic assumptions in IndiGolog are that an action's outcome is correctly modeled, and that all exogenous events beyond the system's control are known. Such assumptions are in conflict with reality, as actions and their models are error-prone and sensing results might be noisy and incomplete. Dismissing such details negatively affects a robot's effectiveness in situations where the control system reasons with belief that contradicts reality.

Consider, for instance, some office delivery robot that is to deliver a letter to office C. Assume that the robot believes to have delivered the letter to office C, while in reality it moved to room A instead (and is unaware of that). Besides the failed task, all future courses of actions will be based on this very wrong belief. As a first step, execution monitoring and a strong background model could enable the detection of such inconsistencies, say, when an object would be considered to be in two rooms simultaneously. For such encountered inconsistencies, there might be more than one explanation (failing actions, sensing faults, or exogenous events). Hence, the robot has to reason about such explanations, demanding for belief management over multiple hypotheses.

In this paper, we present an approach of such a belief management system in IndiGolog that follows the history-based diagnosis approach by Iwan [Iwan, 2002]. Both, IndiGolog and history-based diagnosis, are based on the situation calculus [McCarthy, 1963]. Instead of addressing "what is wrong", the type of diagnoses is along the lines of explaining "what happened"[McIlraith, 1999]. Our system allows the agent to detect inconsistencies between the modeled belief and real sensor values, and generates a number of hypotheses to explain the mismatch. While we adopt the most likely hypothesis for immediate operation, we keep a pool of alternatives in case future data prove the favored one to be wrong. We limit the pool size in order to keep the number of explanations manageable. Experimental results from a delivery domain show the capabilities of our approach.

The remainder of this paper is organized as follows. First, we introduce IndiGolog and Iwan's history-based diagnosis approach. In Section 3 we formalize our belief management system and show its integration into IndiGolog. Further, we introduce a pool of possible hypotheses and constitute a min-

[*]Part of the presented work was done while A. Ferrein was affiliated with the Robotics and Agent Research Lab at the University of Cape Town, South Africa.

imum pool size to still be able to explain a maximum number of k faults. Experimental results can be found in Section 4. We conclude with a discussion of related work and an outlook to our future research.

2 Background

2.1 The Situation Calculus

The situation calculus [McCarthy, 1963] is a sorted second order logical language with equality that allows one to reason about actions and their effects. Starting in the initial situation S_0 (when no actions have occurred yet), action sequences (histories) can be evaluated to define the current situation s. For this purpose, there is a special function symbol $do : action \times situation \rightarrow situation$ that denotes the situation s after performing action α in situation s ($s' = do(\alpha, s)$). Properties holding in a situation are represented by so-called fluents, where fluents are functions or relations with a situation term as last argument. Action precondition axioms $\mathcal{D}_{ap}$ of the form $Poss(\alpha(\vec{x}), s) \equiv \Pi_\alpha(\vec{x})$ and effect axioms describe the validity of an action in a situation s and the effects regarding s'. In Reiter's variant of the situation calculus, action effects are axiomatized by so-called successor state axioms $\mathcal{D}_{ssa}$ of the form: $F(\vec{x}, do(\alpha, s)) \equiv \varphi^+(\alpha, \vec{x}, s) \vee F(\vec{x}, s) \wedge \neg\varphi^-(\alpha, \vec{x}, s)$, with formulas $\varphi^{+/-}(\alpha, \vec{x}, s)$ evaluating F to true or false respectively.[1] In combination with foundational axioms Σ, unique name axioms for actions $\mathcal{D}_{una}$, and some axioms about the initial situation $\mathcal{D}_{S_0}$, $\mathcal{D}_{ssa}$ and $\mathcal{D}_{ap}$ form the so-called *basic action theory* $\mathcal{D} = \Sigma \cup \mathcal{D}_{ssa} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$. For more details, we refer the reader to [Reiter, 2001].

2.2 History-based Diagnosis

We follow the approach of [Iwan, 2002], denoting histories with σ, $\bar{\delta}$ or $\bar{\alpha}$, situation terms with σ or s. Sometimes, we use both terms synonymously, but it will be evident from the context. Iwan distinguishes between action variations and exogenous action insertions as explanations for inconsistencies between an observation ϕ (a sensor value) and an action history $\bar{\delta}$ (a situation term). All possible variations and insertions are completely modeled in terms of predicates Var and Ins, where $Var(\alpha, A(\vec{x}), \sigma) \equiv \Theta_A(\alpha, \vec{x}, \sigma)$ and $Ins(\alpha, \sigma) \equiv \Theta(\alpha, \sigma)$ state under which conditions Θ_A and Θ respectively, action A is a variation of action α and some action α is a valid insertion into the action history. We use the notion of executable action histories $Exec(do([\alpha_1, \ldots, \alpha_n], S_0)) \doteq \bigwedge_{j\in\{1,\ldots,n\}} Poss(\alpha_j, do([\alpha_1, \ldots, \alpha_{j-1}], S_0))$[2] and the notion of extended variations, which is inductively defined as:

1. $EVar(\epsilon, \epsilon) \doteq \top$,
2. $EVar(\epsilon, \bar{\alpha}.\alpha) \doteq \bot$,
3. $EVar(\bar{\delta}.\delta, \epsilon) \doteq Ins(\delta, do(\bar{\delta}, S_0)) \wedge EVar(\bar{\delta}, \epsilon)$,
4. $EVar(\bar{\delta}.\delta, \bar{\alpha}.\alpha) \doteq (\,Var(\delta, \alpha, do(\bar{\delta}, S_0)) \wedge EVar(\bar{\delta}, \bar{\alpha})) \vee (Ins(\delta, do(\bar{\delta}, S_0)) \wedge EVar(\bar{\delta}, \bar{\alpha}.\alpha))$

meaning that $\bar{\delta}.\delta$ is an extended variation of $\bar{\alpha}.\alpha$ (or ϵ as in case 3) if δ is a valid variation (or insertion, resp.) of α in situation $do(\bar{\delta}, S_0)$ and $\bar{\delta}$ is an extended variation of $\bar{\alpha}$. The notion $\bar{\delta}.\delta$ refers to the history of actions in $\bar{\delta}$ plus the single action δ. The equivalent notation as a situation term is $do([\bar{\delta}, \delta], s)$. We denote the empty history with ϵ. In the introductory robot example, going to office A is a variation of going to office C, while adding an exogenous event stating that the robot was robbed of the letter represents an insertion. A ground action sequence $\bar{\delta}$ is an explanatory history-based diagnosis for an observation ϕ and a history $\bar{\eta}$ iff $\mathcal{D} \models ExplDiag(\bar{\delta}, \phi, \bar{\eta})$ with $ExplDiag(\bar{\delta}, \phi, \bar{\eta}) \doteq EVar(\bar{\delta}, \bar{\eta}) \wedge Exec(\bar{\delta}) \wedge \phi[\bar{\delta}]$. Note that we assume a finite limit to the number of insertions and variations, which is reasonable as during one execution step usually only a limited number of faults occur. Given a complete model considering all possible faults, all possible inconsistencies can be explained. For ranking diagnoses, we define a preference value, as is discussed in Section 3.1.

2.3 IndiGolog

IndiGolog [Giacomo *et al.*, 2009] is an online variant of Golog [Levesque *et al.*, 1997] accounting for offline projections and sensing. It is a robot programming language where the semantic of its constructs is defined in the situation calculus. To this end, a one-step transition semantic with predicates $Trans$ and $Final$ is defined. A program configuration is a tuple $\langle\gamma, s\rangle$, where γ is a program and s a situation. The special predicate $Trans(\gamma, s, \gamma', s')$ transforms program γ in the situation s into the program γ' resulting in the situation s'. To denote final (terminating) configurations, a predicate $Final(\gamma, s)$ exists. For example, to transform a primitive action, the following $Trans$ and $Final$ predicates are required:

1. $Trans(nil, s, \gamma', s') \equiv \bot$;
2. $Trans(\alpha, s, \gamma', s') \equiv Poss(\alpha, s) \wedge \gamma' = nil \wedge s' = do(\alpha, s)$
3. $Final(nil, s) \equiv \top$
4. $Final(\alpha, s) \equiv \bot$.

The transition for a primitive action is: $\langle\alpha, s\rangle \rightarrow \langle nil, do(\alpha, s)\rangle$. The latter is a final configuration as the predicate $Final(nil, do(\alpha, s))$ holds. Loops, conditionals, and recursive procedures are defined in a similar way (see [Giacomo *et al.*, 2009] for a complete overview).

To account for sensing results, a number of sensing axioms $SF(\alpha(\vec{x}), s) \equiv F(\vec{x}, do(\alpha(\vec{x}), s))$ which relate sensing actions with fluents are introduced. For each non-sensing action α_i, one adds an axiom $SF(\alpha_i, s) \equiv \top$.

To give an example, consider that our robot wants to sense whether a door is open: $SF(senseDoor(d), s) \equiv Open(d, s)$ where $senseDoor$ is a primitive action and $Open$ is a fluent. All sensing axioms are collected in the set $\{Senses\}$. In IndiGolog, one further needs a set $\mathcal{C}$ that contains axioms for reifying programs as terms. See [Giacomo *et al.*, 2009] for a detailed discussion. Throughout the rest of the

[1] For space reasons, we only show relational fluents here. Further note that, as usual, all free variables in formulas are meant to be implicitly universally quantified. We abbreviate the list of variables $x_1, \ldots, x_n$ with $\vec{x}$.

[2] $\Theta_A(a, A(\vec{x}), s)$ and $\Theta(a, s)$ are similar to formulas $\Pi_a(\vec{x})$ in the right-hand side of action precondition axioms. $do([\alpha_1, \ldots, \alpha_n], \sigma)$ is an abbreviation for the term $do(\alpha_n, do(\alpha_{n-1}, \cdots do(\alpha_1, \sigma) \cdots))$.

paper, we hence use an extended basic action theory $\mathcal{D}^* = \mathcal{D} \cup \mathcal{C} \cup \{Sensed\}$.

Having all preliminaries in place, we will show the integration of history-based diagnosis into IndiGolog in the next section.

3 Belief Management in IndiGolog

Our belief management is based on a pool of diagnoses explaining an agent's sensory inputs. In our system, an agent's belief is the set of all fluents' truth values in the current situation, as derived by $\mathcal{D}^*$. For encountered inconsistencies, the agent computes a number of possible hypotheses that explain them. In order to control the space of possible hypotheses, the agent keeps the most preferred hypothesis to explain the just occurred inconsistency. This is very much in the spirit of IndiGolog, where the agent also commits to the next action by directly executing it. In this section, we formally define history-based diagnosis and show the integration into the IndiGolog interpreter.

3.1 History-based Diagnosis with Invariants

Let us start with the notion of consistency. In this context, for one, we need to check whether the current situation is consistent with additional domain related environment invariants. Invariants are represented by the predicate $Invaria(s) \doteq \bigwedge_i \iota_i(s)$, a conjunction of invariants sentences $\iota(s)$ over fluents formulas. An example for an invariant is $\iota_n(s) \equiv [carry\_something \supset \exists o.has\_object(o)]$ stating that if the pressure sensor detects some weight, then the robot is carrying some object. Such invariants are required to detect inconsistencies within not directly accessible fluents. For another, we need to detect inconsistencies between modeled belief and the real world w.r.t. our sensors. We have to compare the modeled belief which is represented by the sense fluent axioms SF with values that come directly from our sensors. To this end, we introduce a helper predicate $RealSense(\alpha, s)$. It is true if the sensor for fluent F connected via the sensing action α returns the value $\top$ (similarly as a sense fluent axiom); and false otherwise. If $SF(\alpha, s) \not\equiv RealSense(alpha, s)$, then the real sensor value contradicts the predicted one, and we have an inconsistency. Due to limited space, we do not introduce $RealSense$ formally, and refer the reader to e.g. [Ferrein, 2008] on how real sensor values can be integrated into IndiGolog.

Definition 1. *A history σ is consistent iff $\mathcal{D}^* \models Cons(\sigma)$ with $Cons(\cdot)$ inductively defined as:*

1. $Cons(\epsilon) \doteq Invaria(S_0)$
2. $Cons(\bar{\delta}.\alpha) \doteq Cons(\bar{\delta}) \wedge Invaria(\bar{\delta}.\alpha) \wedge$
   $[SF(\alpha, \bar{\delta}) \wedge RealSense(\alpha, \bar{\delta}.\alpha) \vee$
   $\neg SF(\alpha, \bar{\delta}) \wedge \neg RealSense(\alpha, \bar{\delta}.\alpha)]$

Before we introduce our diagnosis approach, we need to define a difference measure on action histories.

Definition 2. *Let $cv : situation \times situation \rightarrow \mathbb{R}^+$, $cv(\bar{\delta}, \bar{\alpha}) = v$ with $v \geq 0$ evaluate the difference between a diagnosis $\bar{\delta}$ and a history $\bar{\alpha}$. cv is called the* change value *and is inductively defined as:*

1. $cv(\bar{\delta}, \bar{\delta}) = 0$
2. $cv(\bar{\delta}.\delta, \bar{\alpha}) = cv(\bar{\delta}, \bar{\alpha}) + val(Ins(\delta, \bar{\delta}))$
3. $cv(\bar{\delta}.\delta, \bar{\alpha}.\alpha) = cv(\bar{\delta}, \bar{\alpha}) + val(Var(\delta, \alpha, \bar{\delta}))$

Ins and Var are as in Section 2.2. We assume standard axiomatization of integers together with their standard operations.

The cost function val assigns a positive value that reflects the impact of repair to each single action insertion and variation. The lump sum of total costs for all variations and insertions yields the final difference value between a diagnosis and a history, and therefore defines the diagnosis' change value. Thus the higher the value, the less plausible a diagnosis is. The assumption that diagnoses representing a lower cardinality of faults are more plausible is commonly used in ranking diagnoses [de Kleer and Williams, 1987].

A history-based diagnosis is an executable extended variation that is consistent with the current sensing history.

Definition 3. *Let $\bar{\delta}$ and σ be histories. Let $Diag(\bar{\delta}, \sigma, v) \doteq EVar(\bar{\delta}, \sigma) \wedge Exec(\bar{\delta}) \wedge Cons(\bar{\delta}) \wedge v = cv(\bar{\delta}, \sigma)$, denoting that $\bar{\delta}$ is an extended variation of σ that is executable and consistent. $\bar{\delta}$ is a proper* history-based diagnosis *(or diagnosis for short) based on σ iff $\mathcal{D}^* \models Diag(\bar{\delta}, \sigma, v)$.*

To illustrate this, we come back to our introductory robot example. Assume the robot performed the following consistent action history $\sigma = [pickup(letter), goto(room1), drop(letter)]$. It now performs a sensing action $sense(letter)$ with

$$RealSense(sense(letter), \sigma) \not\equiv SF(sense(letter), \sigma)$$

which is inconsistent. Possible explanations are: (1) the robot failed to pick it up: $\bar{\delta} = [pickupFail, goto(room1), dropNothing]$); (2) the sensing result was wrong: $RealSense(sense(letter), \sigma_2) \equiv SF(sense(letter), \sigma_2)$; or (3) somebody snatched the letter: $\bar{\delta}' = [pickup(letter), snatch(letter), goto(room1), dropNothing]$.

We can estimate an upper bound for the size of the diagnosis space w.r.t the number of variations and insertions that happened between two consecutive actions.

Theorem 1. *Let l be the length of the history σ, n be the number of different possible exogenous events, k be the maximum number of insertions between two actions of σ, and m be the maximum number of variations of an action. Then, the number H of potential diagnosis candidates is $H = ((m+1) \cdot \sum_{i=0}^{k} n^i)^l$.*

Proof (Sketch). For every action there are m variations plus the action itself. Every action can be followed by 0 to k insertions. The number of insertions is exponential in i. The whole term is exponential in the history length l. □

To accommodate a possible state-space explosion, in our implementation we keep a fixed-sized pool *Pool* of diagnoses together with their respective change values. As we assume a finite number of insertions and variations, we can define the pool of diagnoses as a formula over all possible diagnoses in case the current history is not consistent with our observations of what happened in the real world. Otherwise, the pool consists of the history itself.

Definition 4. *Let σ be a history. Then Pool is defined as*

$$\begin{aligned}&Pool(\sigma) = ((\bar{\delta}_1, v_1), \cdots, (\bar{\delta}_n, v_n)) \doteq \\ &\quad Cons(\sigma) \land \bar{\delta}_1 = \sigma \land v_1 = 0 \land \cdots \land \bar{\delta}_n = \sigma \land v_n = 0 \lor \\ &\quad \neg Cons(\sigma) \land Diag(\bar{\delta}_1, \sigma, v_1) \land \cdots \land Diag(\bar{\delta}_n, \sigma, v_n)\end{aligned}$$

The implementation of our belief management system evaluates the pool of hypotheses in a lazy fashion. It focuses therefore on (one of) the simplest explanation(s) for an inconsistency first, that is a hypothesis with the smallest preference value.

Definition 5. *Let σ be a history. The preferred diagnosis prefDiag is defined as:*

$$\begin{aligned}&prefDiag(\sigma) = \bar{\delta} \doteq \\ &\quad \exists \bar{\delta}_1, v_1, \ldots, \bar{\delta}_n, v_n . Pool(\sigma) = ((\bar{\delta}_1, v_1), \cdots, (\bar{\delta}_n, v_n)) \land \\ &\quad [\bar{\delta} = \bar{\delta}_1 \land v_1 \leq v_2 \land \cdots \land v_1 \leq v_n \lor \\ &\quad \bar{\delta} = \bar{\delta}_2 \land v_2 \leq v_1 \land v_2 \leq v_3 \land \cdots \land v_2 \leq v_n \lor \\ &\quad\quad \vdots \\ &\quad \bar{\delta} = \delta_n \land v_n \leq v_1 \land \cdots \land v_n \leq v_{n-1}]\end{aligned}$$

For a specific number c of faults we can determine a necessary minimum pool size (i.e. the length of the formula) ensuring the completeness of our approach. We can show that w.r.t. a fixed maximum number of changes c to a given history, the required pool size is much smaller than the number H established in Theorem 1.

Theorem 2. *Let σ be a history and p be number of diagnoses of $Pool(\sigma)$. Let c be the maximum number of all insertions i and variations v to a history σ and let k, l, m and n be as in Theorem 1. Further, let $\tau = \sum_{c'=1}^{c} \sum_{i=0, v=c'-i}^{c'} \binom{l}{v} m^v \binom{i+l-1}{i} n^i$. If $c \leq k, l$ then τ is the exact amount of possible hypotheses. τ is an upper bound for $c > k, l$. With $p \geq \tau$ we can guarantee that our approach is complete.*

Proof (Sketch). We investigate variations v and insertions i separately, where the product of the corresponding options determines the total amount of diagnoses. For each of the $\binom{l}{v}$ possibly faulty action combinations we have m^v instances. Regarding insertions, after adding i elements to σ we get $|\sigma'| = i + |\sigma| = i + l$. As the first element is fixed, similarly as with the variations we have for each of the $\binom{i+l-1}{i}$ combinations n^i instances. Consequently, we have to sum over all different distributions of variations and insertions and finally sum over all $c' \leq c$. If the pool size is greater or equal to this maximum number of hypotheses, then obviously all hypotheses can be considered to be in the pool. □

Algorithm 1: Extended mainloop of the Prolog Implementation of IndiGolog.

1. *Exogenous events*

```
indigo(E,H) :-  exog_occurs(Act),
 exog_action(Act),!,indigo(E,[Act|H]).
```

2. *Transition of the program*

```
indigo(E,H):-
  readPool(PrefD,H),
  transBM(E,H,E1,H1,PrefD),
  storePool(PrefD,H1),
  indixeq(H,H1,H2),!,indigo(E1,H2).
```

3. *Final condition check*

```
indigo(E,H) :- final(E,H), length(H,N),
 write(N), write(' actions.'), nl.
```

3.2 Integrating the Diagnosis Step into IndiGolog

Now, we show the integration of our diagnosis step into IndiGolog. The Prolog implementation of the mainloop of our extended IndiGolog interpreter (cf. [Giacomo *et al.*, 2009] for a complete description of the implementation) is shown in Alg. 1. First it is checked, whether an exogenous event occurred. If so, it is entered up front in the action history. Step 3 of the mainloop checks, if the program reached a final configuration and if so, the interpreter terminates. Step 2 of Alg. 1 performs a single-step transitions of the program. Here, we integrated the diagnosis step. Therefore, we exchanged the original *Trans* predicate with a predicate *TransBM* that is introduced below; it extends Step 2 with a consistency check and the pool update (as described in the previous section). Before we can check for consistency, we need to get the pool of diagnoses from Prolog's database (via the predicate `readPool`); after our consistency check, the pool is stored again (`storePool`). In our implementation, we incrementally generate the different hypotheses. This means that we do not compute all possible hypotheses after the execution of each action. Only in case of an inconsistency, we extend invalid hypotheses in the pool. This is done for run-time efficiency. The predicate `indixeq` is also used in the original version of the interpreter and provides the interface to execute actions in the real world. Now, we come to the consistency check step and define *TransBM* formally:

$$\begin{aligned}&TransBM(\gamma, s, \gamma', s') \equiv \\ &\quad \exists \gamma'' . Trans(\gamma, s, \gamma'', s') \land \exists \bar{\delta} . prefDiag(s') = \bar{\delta} \land \\ &\quad \exists \gamma''' . align(\bar{\delta}, \gamma''') \land \gamma' = \gamma'' . \gamma'''\end{aligned}$$

TransBM takes a program γ in situation s and transforms it to program γ' in situation s'. First, the *Trans* predicate, which is defined as usual (cf. [Giacomo *et al.*, 2009]) performs a single step transition. *prefDiag* identifies the new preferred diagnosis $\bar{\delta}$, which is used to generate a new program consisting of necessary actions to retrieve a consistent situation.

Assuming a correct model and a sufficient pool size, now we have at least one consistent diagnosis that explains the current world state. However, the situation s' could still be inconsistent, and future program transitions might not be possible or will lead to more inconsistencies. Therefore, we need to generate a consistent world state. One possibility could be to just go on with the program execution on history $\bar{\delta}$. But as this is only a hypothesis, it does not seem advisable to do so. Instead, we decided to introduce a predicate $align$ that takes the preferred diagnosis $\bar{\delta}$ and generates a program γ''' whose world state agrees with $\bar{\delta}$. This is achieved by a special action $set\_val(F(\vec{x}, \psi_F))$ that sets the value of fluent F to the value ψ_F explicitly (see [Ferrein, 2008] for details on how to define such an action in IndiGolog). $align(\bar{\delta}, \gamma)$ generates a sequence of "set" actions, one for each fluent in the domain description. To this end, we need a number of macros of the form $\psi_{F_1}(\vec{x_1}, s) \doteq F_1(\vec{x_1}, s), \ldots, \psi_{F_n}(\vec{x_n}, s) \doteq F_n(\vec{x_n}, s)$, one for each fluent. $align$ is defined as:

$$\begin{aligned} &align(\bar{\delta}, \gamma) \doteq \\ &\qquad \gamma = set\_val(F_1(\vec{x_1}), \psi_{F_1}(\vec{x_1}, \bar{\delta})); \ldots; \\ &\qquad\quad set\_val(F_n(\vec{x_n}), \psi_{F_n}(\vec{x_n}, \bar{\delta})). \end{aligned}$$

The execution of this sequence of set_val actions leads to a world state that agrees with $\bar{\delta}^*$. In the next section we show the results of our belief management.

4 Experimental Results

In order to evaluate our approach, we simulated and analyzed two robot control systems for delivery robots. We compared one system (**BM**) equipped with our belief management system against a plain one (**P**) that lacks it. The contestants were to move 3 objects in an office environment consisting of 59 rooms and 12 hallway segments (resembling our department). The robots' set of actions allowed them to pickup an object, move between connected rooms, and drop an object.

For our tests, we defined three action fault types and one exogenous one: The robot might (1) fail to pick up an object $Var(pickup(\cdot), pickupNothing, s)$, (2) pick up the wrong object $Var(pickup(o_1), pickup(o_2), s) \equiv obj(o1) \wedge obj(o_2) \wedge o_1 \neq o_2)$, with $obj(o) = o \doteq letter \vee o = box \vee \ldots$ defining a finite set of objects; and (3) fail to release an object $Var(drop(o), dropNothing, s) \equiv has\_obj(o, s)$. These faults resemble issues that frequently occur in reality. As exogenous issue, mean agents snatch objects currently carried by the robots $Ins(snatchObject(o), s) \equiv has\_obj(o, s))$.

Our four fault scenarios **F1** to **F4** define the following fault probabilities: In **F1** the probability for picking up the wrong object is 0.2 and 0.4 for it to fail entirely. Additionally, in **F2** a 0.3 probability for a failing drop action is defined. In **F3** we add a probability of 0.05 that a sensing action fails, extended by mean agents snatching objects with a probability of 0.2 in **F4**. In order to model real robots more closely, we implemented continuous passive sensing in IndiGolog[3], and defined three sensing rates **S1** to **S3**: **S1** refers to the availability of sensor data after every action. For **S2** sensor data is available after every second action and for **S3** after every third one. The sensing rates represent the fact that due to resource limitations (e.g. bandwidth, power, or computation time), in reality sensing might not be available at high rates.

[3]It is quite common in real robot systems that they automatically provide sensing information after each action execution without actively triggering special sensing actions.

For the results given in Table 1(a), the contestants **P** and **BM** had to prove themselves in the context of 50 different missions (plans). For any scenario and sensing rate they had to solve these missions 10 times with varying seeds. In order to exclude any impact of planning issues on the results, both agents used a simple control program generating a trace of typical 40 actions in the absence of faults. The table reports the rate of successfully completed missions. A mission is failed if its execution is aborted or exceeds a timeout of 120 minutes, or if the 3 objects are not at the desired locations.

The results in Table 1(a) show that the agent **BM** clearly outperforms the plain one with a significance level of 1%. Though the success rate decreases along decreasing sensor and increasing fault rates, in contrast to the plain agent, **BM** is still able to finish most of its missions. Especially the impact of sensing rates can be minimized using the agent **BM** instead of the plain one.

In Table 1(b) we report the average runtime for **BM** for all twelve scenarios[4]. The average runtime of the plain agent is almost constant at 29.4 seconds over all scenarios. Whereas there is just a very small overhead of the BM agent in F1, F2 and F3 the overhead in F4 increases exponentially.

5 Discussion

In this paper we presented a belief management system that allows an agent or robot to detect inconsistencies between its internal representation of the world and reality. In order to be able to cope with such inconsistencies, the system provides explanations about what has happened in terms of the occurrence of wrong outcomes of primitive and sensing action and exogenous events the agent is not aware of. This is in contrast to diagnoses that answer the question of "*what is wrong*" and was first addressed in the situation calculus by [McIlraith, 1999] (later also by [Sohrabi *et al.*, 2010]). McIlraith addresses explanatory diagnoses which are continuations of a given history of actions. It is shown that finding such diagnoses is analogous to planning. [Grastien *et al.*, 2007] established the connection to satisfiability problems.

The main contributions of this paper are a formalization of the belief management system based on the situation calculus and history-based diagnosis, and the integration of the system into an existing IndiGolog interpreter. Besides a formal proof that the system (under some assumptions) is capable to find the right explanation for an inconsistency, we present experimental results from a simulated delivery robot domain. These show that an agent equipped with the proposed system is able to finish more tasks successfully under the occurrence of faults and exogenous events. We integrated the diagnosis step in a fashion similar to the work of De Giacomo et al. [de Giacomo *et al.*, 1998; de Leoni *et al.*, 2007] about execution monitoring. While they show how to integrate a monitoring step in the IndiGolog interpreter, they do not use a sophisticated diagnosis mechanism. Other related works on execution monitoring in Golog is e.g. [Lespérance and Ng, 2000; Ferrein *et al.*, 2004].

[4]On an Intel quad-core CPU with 2.40GHz and 4GB of RAM

	Scenario							
	F1		F2		F3		F4	
	P	BM	P	BM	P	BM	P	BM
S1	100 / 0	100 / 0	51 / 15	100 / 0	18 / 10	96 / 7	6 / 11	77 / 14
S2	38 / 27	100 / 0	24 / 13	100 / 0	15 / 7	80 / 15	4 / 5	65 / 17
S3	31 / 20	99 / 5	20 / 10	99 / 2	17 / 9	71 / 14	3 / 5	52 / 22

(a) Percentage of successfully finished missions plus standard deviations.

	Scenario			
	F1	F2	F3	F4
S1	38 / 7	44 / 8	75 / 25	845 / 444
S2	48 / 10	56 / 11	83 / 22	644 / 190
S3	57 / 27	67 / 16	87 / 21	771 / 237

(b) Average runtime in seconds (BM) and standard deviations.

Table 1: Results for scenarios F1 - F4 and sensing rates S1 - S3 for a robot using belief management (BM) and a plain one (P).

A drawback of the approach is its current high complexity, as the number of potential explanations to be considered is double exponential in the number of faults considered and the length of the action history. Even if pruning techniques help to control this number, this potentially limits the applicability to more complex domains. In future work we will investigate whether our proposed system can benefit from more compact representations already used in the planning, diagnosis or model checking domain. Another important issue is that all potential faults and their effects have to be modeled. It is still questionable if and how such models can be generated automatically in an efficient way, and if there is a way to cope with non-modeled faults.

Acknowledgments

This work has been partly funded by the Austrian Science Fund (FWF) under grants P22690 and P22959. A. Ferrein is supported by a grant of the Alexander von Humboldt foundation as a Feodor Lynen fellow. We would like to thank the anonymous reviewers for their helpful comments and M. Thielscher for his help to revise an earlier version of this paper.

References

[de Giacomo *et al.*, 1998] G. de Giacomo, R. Reiter, and M. Soutchanski. Execution monitoring of high-level robot programs. In *Proc. KR*, pages 453–465, 1998.

[de Kleer and Williams, 1987] Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[de Leoni *et al.*, 2007] M. de Leoni, M. Mecella, and G. de Giacomo. Highly dynamic adaptation in process management systems through execution monitoring. In *Proc. BPM'2007*, pages 182–197. Springer, 2007.

[Ferrein *et al.*, 2004] A. Ferrein, Ch. Fritz, and G. Lakemeyer. Online decision-theoretic golog for unpredictable domains. In *Proc. KI-2004*, LNCS, pages 322–336. Springer, 2004.

[Ferrein, 2008] A. Ferrein. *Robot Controllers for Highly Dynamic Environments with Real-time Constraints*. PhD thesis, Knowledge-based Systems Group, RWTH Aachen University, Aachen Germany, 2008.

[Giacomo *et al.*, 2009] G. De Giacomo, Y. Lespérance, H. J. Levesque, and S. Sardina. *Multi-Agent Programming: Languages, Tools and Applications*, chapter IndiGolog: A High-Level Programming Language for Embedded Reasoning Agents, pages 31–72. Springer, 2009.

[Grastien *et al.*, 2007] Al. Grastien, Anbulagan, J. Rintanen, and E. Kelareva. Diagnosis of discrete-event systems using satisfiability algorithms. In *Proc. AAAI-07*, 2007.

[Iwan, 2002] G. Iwan. History-based diagnosis templates in the framework of the situation calculus. *AI Communications*, 15(1):31–45, 2002.

[Lespérance and Ng, 2000] Y. Lespérance and H.-K. Ng. Integrating planning into reactive high-level robot programs. In *Proc. CogRob*, pages 49–54, 2000.

[Levesque *et al.*, 1997] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl. GOLOG: A logic programming language for dynamic domains. *The Journal of Logic Programming*, 31(1-3):59 – 83, 1997.

[McCarthy, 1963] J. McCarthy. Situations, Actions and Causal Laws. Technical report, Stanford University, 1963.

[McIlraith, 1999] S. McIlraith. Explanatory diagnosis: Conjecturing actions to explain observations. In *Logical Foundations for Cognitive Agents: Papers in Honour of Ray Reiter*, pages 155–172. Springer, 1999.

[Reiter, 2001] R. Reiter. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

[Sohrabi *et al.*, 2010] S. Sohrabi, J. A. Baier, and S. A. McIlraith. Diagnosis as planning revisited. In *Proc. KR*, pages 26–36, 2010.