Knowledge-Aware Execution of Programs in IndiGolog

Clemens Mühlbacher and **Gerald Steinbauer**[1]

Abstract.

Agents that act in dynamic environments often face situations where their belief about the world is in contradiction with reality. The lack of secured knowledge may let the agents perform wrong or even dangerous actions. In order to deal with this two different approaches can be used. One approach limits the reasoning to that knowledge that is secure while the other approach actively diagnose and repair problems in the knowledge. In this paper we briefly discuss how the two approaches can be combined in order to improve the robustness of execution of high-level IndiGolog programs controlling an agent.

1 Introduction

An autonomous agent must deal with the dynamics in its environment. This is especially challenging as the world often does not evolve as the agent expects. For example consider a simple robot which patrol between several rooms in an office environment. Due to the nature of the office environments the doors can be opened or closed from humans without notifying the agent. Thus the agent needs to be capable to deal with a situation that the door is closed but the agent has detected that the door was open 10 minutes ago. But the environment can also change in an unexpected way if an action does not result in the effect the agent is expecting. For example if the agent wants to pick up an object but the object was too slippy. Thus the agent does not hold the object in its hand after the pick up action.

Many different approaches where proposed to deal with changes in the environment; see for example [6], [8] or [12]. Many of those approaches use a ranking to specify what happens with some likelihood. It is very common in many approaches to specify the ranking in such a way that the recent sensing outcome is considered to be correct in the best ranked alternatives. Such an approach seems to be promising to robustly fulfill a task [18]. But it can result in spurious results as the approach assume trustworthy sensors. For example consider an agent which senses if a door is closed every 10 minutes. If the door is opened after some time the agent can reason that either someone has opened the door or that the sensing was wrong. If the agent would just use the newest sensing information the agent would belief the door is opened even in a case the sensing was wrong. Thus the agent may raise an alarm without a reason. If the agent track both hypotheses and try to find the hypothesis which is more likely the agent can observe the door again. Thus the agent could overcome a faulty measurement.

To tackle the problems arising from a dynamic world we propose an extension of the situation calculus [16]. To deal with the unexpected changes in the environment the agent performs a diagnosis on the history of performed actions. As a result the diagnosis may end up in multiple hypothesis. If the diagrement on the agents belief in the hypothesis is too big to achieve the goal the agent performs actions to acquire knowledge about the world and to rule out different hypothesis. This extension of the situation calculus was integrated in the high-level control language IndiGolog [9]. This makes it possible to use the extended situation calculus to control an agent to achieve a task more robustly. We expect that these abilities will result in an agent high-level control which is more autonomic and robust then other control approaches.

The remainder of the paper is organized as follows. In Section 2 we will discuss the prerequisites of our approach. In the proceeding section we will define how the extended situation calculus can be used within the transition semantics of IndiGolog. In Section 4 we will show the impact of the changes on IndiGolog transition semantics. Before we conclude the paper we will give a short overview of related research in Section 5. Finally we will conclude the paper and point out some future work.

2 Prerequisites

As a formal base we use the situation calculus [16]. The situation calculus is a second order language with equality. The situation calculus use fluents to specify predicates which can change over time. To specify how the world evolves over the time a situtation term is used. The sitution term is defined through the function $do : action \times situation \rightarrow situation$. The function $s' = do(\alpha, s)$ specifies that the agent performs an action α in situation s which result in the situation s'[2]. The action preconditions are specified through $Poss(\alpha(\vec{x}), s) \equiv \Pi_{\alpha(\vec{x})}$. The preconditions of all actions are specifed in the set $\mathcal{D}_{ap}$. Beside the precondition also the effect of an action needs to be specified. To define the effect on the fluents successor state axioms are used. The successor state axioms are defined as $F(\vec{x}, do(\alpha, s)) \equiv \varphi^+(\alpha, \vec{x}, s) \vee F(\vec{x}, s) \wedge \neg\varphi^-(\alpha, \vec{x}, s)$, where F is a fluent. φ^+ specifies the condition under which an action triggers the fluent to hold in the next situation. φ^- specifies the condition under which an action trigger the fluent to become false in the next situation. Each of the successor state axioms is defined for a fluent and is stored in the set $\mathcal{D}_{ssa}$. Beside the successor state axioms which model the dynamics of the world the sitution calculus use also the constant S_0 to represent the initial situation and the set $\mathcal{D}_{S_0}$ to specify which fluents holds in the initial situation. To complete the reasoning about the world the situation calculus uses some foundation axioms Σ and unique name axioms for actions $\mathcal{D}_{una}$. The combination of all these axioms define the basic action theory $\mathcal{D}_{ap}$, $\mathcal{D}_{ssa}$, $\mathcal{D}_{S_0}$, Σ and $\mathcal{D}_{una}$ [19].

[1] The authors are with the Institute for Software Technology,Graz University of Technology, Graz, Austria, {cmuehlba,steinbauer}@ist.tugraz.at. The work has been partly funded by the Austrian Science Fund (FWF) by grant P22690.

[2] We will use $do([\alpha_1, \ldots, \alpha_n], \sigma)$ as an abbreviation for the term $do(\alpha_n, do(\alpha_{n-1}, \cdots do(\alpha_1, \sigma) \cdots))$.

To gather information about the world the agent must be able to perform sensing actions. To specify the effect of a sensing action the modified situation calculus proposed in [4] is used. The situation calculus is modified in such a way that the agent stores a history of actions together with the sensing results instead of a pure situation term. To reason about this it is spitted in a situation term containing the executed actions and the set $\{Sensed\}$ which gather the information about the sensing. To specify the information which is a result of the sensing the predicate $SF(\alpha, s) \doteq \phi(s)$ is used. The predicate specifies the effect the sensing action α can have in the situation s. If the sensing action return true as a result of the sensor reading $\phi(s)$ is asserted to hold in the situation s. If the sensing action return false as a result of the sensor reading $\neg\phi(s)$ is asserted to hold in the situation s. Due to the asserting of a formula the situation can become contradicting if the sensor reading is in contradiction to the effects the actions have if executed. If an agent repeatedly sense its environment in a situation already one wrong measurement could result in a contradicting situation.

To deal with such contradicting situations a diagnosis approach was proposed in [11]. The diagnosis try to alter the situation in such a way that the situation becomes consistent with the sensor reading. To alter a situation two predicates are used. The first predicate $Varia(\alpha(\bar{x}), \alpha'(\bar{x}), s') \doteq \Theta_{\alpha'}(\alpha, \bar{x}, s')$ specifies under which condition α' is a valid variation of α[3]. The second predicate $Insert(\alpha', s') \doteq \Theta$ specifies under which condition action α' is a valid insertion into the situation s'.

The diagnosis process was further extended in [10] to use background knowledge $BK(s)$. This background knowledge is used to state the invariants of the world the agent is acting in. For example the background knowledge can state that an object is only at one place at a certain point in time. Using this background knowledge the agent can detect situation which are not possible in the world. Thus physical impossible situations can be detected. This is used to prune the possible variations of a situation to those situation which are consistent with the constraints of the world.

Furthermore in [10] a ranking was proposed to use only those alternatives which are the most likely one. The ranking ins designed in such a way that a smaller number of faulty actions is preferred. Thus the ranking follows the principle of ocam's razor. The lowest ranked situation alternations are stored in the so called pool $W(s)$. We will use this pool to reasoning about what the agent knows for sure.

To use the sensing capabilities together the execution of actions the high-level control language IndiGolog was proposed in [9]. This high-level control language uses a transition semantic specified through the predicate $Trans(\delta, s, \delta', s')$ to specify that a program δ can evolve in situation s into program δ' with the resulting situation s'. Additionally a predicate $Final(\delta, s)$ is used to specify that the remaining program can legally terminate in situation s. To perform reasoning about the predicates $Trans$ and $Final$ an extended basic action theory is used. The extended basic action theory consist of the axioms from the basic action theory combined with the axioms in $\{Sensed\}$ and the IndiGolog program reefied as terms $\mathcal{C}$. Formally stated as $\mathcal{D}^* = \Sigma \cup \mathcal{D}_{ssa} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \cup \mathcal{C} \cup \{Sensed\}$.

3 Program Transitions with Knowledge Management

To perform transitions according to the knoweldge of the agent we need first to define the knowledge of the agent. As defined in [10] we define the pool of preferred diagnosis $W(s)$ as those consistent alternative situation which have the best ranking. The different situation in the pool are used to describe what is known by the agent for sure. In order to restrict the decision making to use only that knowledge that is secure we follow an epistemic approach. Like in [7] we define the knowledge the agent knows for sure as that knowledge that holds in all possible situations. Therefore we define the predicate $Knows(\phi, s) \equiv \forall s' \in W(s).\phi(s')$. If we restrict reasoning to that knowledge possibly dangerous situations can be avoided. It is common to use only secured knowledge to progress IndiGolog program execution [14, 9]. The advantage of these approaches is that programs are only progressed if the agent possesses enough secured knowledge. The drawback is that if not enough knowledge is available the program is aborted.

In order to deal with this problem we propose to combine the epistemic program execution with the diagnosis approach for situations. Because diagnosis provides evidence about the secure knowledge we will not stop execution once not enough knowledge is available. In contrast we suggest to perform an active diagnosis step in order to obtain an enrichment of the secure knowledge possibly allowing the agent to continue.

In order to be able to use knowledge management in program transition we adopt the semantics from IndiGolog in order to reflect the robot's secured knowledge. We redefine the predicate $Trans$ which specifies how a program δ progresses in a situation s. For instance the robot has to know that an action's precondition is fulfilled in order to guarantee safe execution. We will omit those transition predicates where the old predicate was simply replaced by the new one. The following list depicts the changed statements:

1. $Trans(\alpha, s, nill, do(\alpha, s)) \equiv Knows(\Pi_\alpha, s)$
2. $Trans(\phi?, s, nill, s) \equiv Knows(\phi, s)$
3. $Trans(\mathbf{if}\ \phi\ \mathbf{then}\ \delta_1\ \mathbf{else}\ \delta_2\ \mathbf{endif}, s, \delta', s') \equiv Knows(\phi, s) \wedge Trans(\delta_1, s, \delta', s') \vee Knows(\neg\phi, s) \wedge Trans(\delta_2, s, \delta', s')$
4. $Trans(\mathbf{while}\ \phi\ \mathbf{do}\ \delta\ \mathbf{endWhile}, s, \delta', s') \equiv \exists\gamma : \delta' = \gamma;\mathbf{while}\ \phi\ \mathbf{do}\ \delta\ \mathbf{endWhile} \wedge Knows(\phi, s) \wedge Trans(\delta, s, \gamma, s')$

The second part of the transition semantics of IndiGolog is the $Final$ predicate. We will omit those transition predicates where the old predicate was simply replaced by the new one. The following list shows the changed statements:

1. $Final(\mathbf{if}\ \phi\ \mathbf{then}\ \delta_1\ \mathbf{else}\ \delta_2\ \mathbf{endif}, s) \equiv Knows(\phi, s) \wedge Final(\delta_1, s) \vee Knows(\neg\phi, s) \wedge Final(\delta_2, s)$
2. $Final(\mathbf{while}\ \phi\ \mathbf{do}\ \delta\ \mathbf{endWhile}, s, \delta', s') \equiv Knows(\neg\phi, s) \vee Final(\delta, s)$

Unfortunately these definitions are not enough to prevent an agent from ending up in a situation where due to little knowledge the agent can't definitely decide if a transition can be taken or the program is final. For instance not knowing if an action's precondition holds does not necessarily mean that the program composed of just that action should be aborted. In order to be sure we have to check also if the agent definitely knows that the precondition does not hold. Otherwise the agent has not enough knowledge to decide. In order to safely decide the program execution we use the decision tree depicted in Figure 1. The tree has three final states (marked by bold solid circles) where the robot is able to safely decide if the program terminates, progresses or aborts.

In order to safely decide we define the predicates $KTrans(\delta, s)$ and $KFinal(\delta, s)$ that specify if an agent is definitely able to decide if a transition is possible respectively if the program can terminate legally. For instance the execution first checks if the agent possesses

[3] We will omit the parameters in the remaining of the paper for readability.

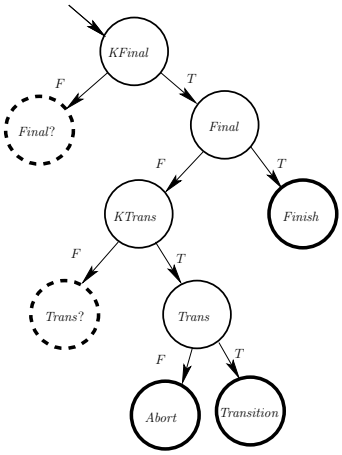

Figure 1. Decision tree for one step transition of a program. Bold leaves depict states where the agent possesses enough knowledge to safely decide program progression. Dashed states are those where no safe program progression can be done.

enough knowledge to safely decide if the program terminates. If this is not the case the agent is facing state $Final?$ where no decision can be made for now. Similar paths exist for the transition as well. Going that way the agent may end up in the state $Trans?$ where no transition can be made for now due to lack of knowledge.

The predicate $KTrans(\delta, s)$ is defined as follows:

1. $KTrans(nil, s) \equiv \top$,
2. $KTrans(\alpha, s) \equiv Knows(\Pi_\alpha, s) \vee Knows(\neg\Pi_\alpha, s)$,
3. $KTrans(\phi?, s) \equiv Knows(\phi, s) \vee Knows(\neg\phi, s)$,
4. $KTrans(\delta_1; \delta_2, s) \equiv KTrans(\delta_1, s) \vee Final(\delta_1, s) \wedge KTrans(\delta_2, s)$,
5. $KTrans(\delta_1|\delta_2, s) \equiv KTrans(\delta_1, s) \vee KTrans(\delta_2, s)$,
6. $KTrans(\pi v.\delta, s) \equiv \exists x KTrans(\delta_x^v, s)$,
7. $KTrans(\delta^*, s) \equiv KTrans(\delta, s)$,
8. $KTrans(\mathbf{if}\ \phi\ \mathbf{then}\ \delta_1\ \mathbf{else}\ \delta_2\ \mathbf{endif}, s) \equiv Knows(\phi, s) \wedge KTrans(\delta_1, s) \vee Knows(\neg\phi, s) \wedge KTrans(\delta_2, s)$,
9. $KTrans(\mathbf{while}\ \phi\ \mathbf{do}\ \delta\ \mathbf{endWhile}, s) \equiv Knows(\phi, s) \wedge KTrans(\delta, s) \vee Knows(\neg\phi, s)$

Predicate $KFinal(\delta, s)$ is defined as follows:

1. $KFinal(nill, s) \equiv \top$,
2. $KFinal(\alpha, s) \equiv \top$,
3. $KFinal(\phi?, s) \equiv \top$,
4. $KFinal(\delta_1; \delta_2, s) \equiv KFinal(\delta_1, s) \wedge KFinal(\delta_2, s)$,
5. $KFinal(\delta_1|\delta_2, s) \equiv KFinal(\delta_1, s) \vee KFinal(\delta_2, s)$,
6. $KFinal(\pi v.\delta, s) \equiv \exists x KFinal(\delta_x^v, s)$,
7. $KFinal(\delta^*, s) \equiv \top$,
8. $KFinal(\mathbf{if}\ \phi\ \mathbf{then}\ \delta_1\ \mathbf{else}\ \delta_2\ \mathbf{endif}, s) \equiv Knows(\phi, s) \wedge KFinal(\delta_1, s) \vee Knows(\neg\phi, s) \wedge KFinal(\delta_2, s)$,
9. $KFinal(\mathbf{while}\ \phi\ \mathbf{do}\ \delta\ \mathbf{endWhile}, s) \equiv Knows(\phi, s) \wedge KFinal(\delta, s) \vee Knows(\neg\phi, s)$

Using $KTrans$ and $KFinal$ we can decide if an agent is in a state where it is facing a lack of knowledge to decide termination or transition. In order to allow the agent to deal with such states we perform active diagnosis. The idea is to execute an action or a sequence of actions that increases the agent's knowledge in a way that a decision about the execution of the original program becomes possible.

To specify active diagnosis program we use the predicate $ActiveDiganosis(\delta, s, \delta^*)$. The process generates a program δ^* that if executed in situation s gives the agent enough information to safely decide about the progression of the original program δ. Please note that for δ^* the new transition predicates apply as well. This ensures that the actions can be safely executed as well. If such a program cannot be found or cannot be safely executed the program execution is aborted due to lack of secured knowledge.

Basically finding the active diagnosis δ^* can be formalized as planning problem. In connection to the history-based diagnosis approach we follow a different way. The predicate $Knows(\phi, s)$ is defined in the way that ϕ has to hold in all consistent situation alternations related to s. In general if we reduced the number of consistent situations the knowledge will increase. This is the result of the definition of knowledge which is formed by the intersection of the fluent sets of the different consistent situations. Using a sequence of actions $\mathbf{a} = [a_1, ..., a_n]$ containing at least one sensing action we are able to prune a possible situation $W(s)'$ not consistent with $\mathbf{a}$. To determine which action or sequence of action will maximize the agent's knowledge we follow the idea of measurement selection for diagnosis presented in [5]. We will select those actions that most likely reduces the number of potential situations. Please note that the selection of promising sequences is based on an probabilistic estimation since sensing results cannot be predicted.

Once the active diagnosis step had been performed the execution of the original program δ continues. Because the secured knowledge has most likely increased the chance to securely progress the program has increased as well.

Putting the predicate for the state transitions $Trans$, $ActiveDiganosis$ and for the state recognition $KTrans$, $KFinal$ together we can define the transition of a Indi-Golog program.

Formally the transition predicate for the belief management is defined as follows: $TransK(\delta, s, \delta', s') \doteq [(\neg KFinal(\delta, s) \vee \neg KTrans(\delta, s)) \wedge ActiveDiganosis(\delta, s, \delta^*) \wedge \delta' = \delta^*.\delta \wedge s' = s] \vee [KTrans(\delta, s) \wedge Trans(\delta, s, \delta', s')]$.

The first part of the transition ensures that the active diagnosis process will be performed if the knowledge of the agent is to limited to reason if a final state is given or a transition can be taken. The second part of the transition perform the transition according to the program semantic if the belief is sufficient to decide which transition to take. This definition poses some non-deterministic choice how the transition is performed. The non-determinis comes into play if $\neg KFinal(\delta, s)$ and $KTrans(\delta, s)$ holds. This opens up the possibility for an implementation to prefer the transition of the program over the active diagnosis calculation.

With the predicate $KTrans(\delta, s)$ and $KFinal(\delta, s)$ we can define the final predicate for the belief management as follows: $FinalK(\delta, s) \doteq KFinal(\delta, s) \wedge Final(\delta, s)$.

This definition ensures that the agent only terminates with a given program if the program is known to be final.

To perform planing in the offline execution of a Indi-Golog program we use the predicate $Do(\delta, s, s')$ which was proposed for Golog [15], ConGolog [2] and Indi-Golog [3]. We use nearly the same definition of this predicate as in Indi-Golog [3] with the only difference that we use $TransK$ instead of the $Trans$ of Indi-Golog. $Do(\delta, s, s') \doteq \exists \delta' : TransK^*(\delta, s, \delta', s') \wedge FinalK(\delta', s')$, where $TransK^*(\delta, s, \delta', s')$ is defined as follows: $TransK^*(\delta, s, \delta', s') \doteq \forall P : [... \supset P(\delta, s, \delta', s')]$. Where ... stands for the conjunctive uni-

versial closure of the following implications:

$$\top \supset P(\delta, s, \delta, s)$$
$$TransK(\delta, s, \delta'', s'') \wedge P(\delta'', s'', \delta', s') \supset P(\delta, s, \delta', s')$$

4 Effects of the Program Transitions with Knowledge Management

Before we discuss some related research we will briefly state the theoretical relation between an ordinary Indi-Golog transition semantics and our transition semantics. Furthermore, we will show the difference on an illustrative example.

As we propose an alternative transition system for Indi-Golog we want to show that our transition semantics will always make a transition if the original Indi-Golog transition semantics augmented with knowledge would make a transition. To show this we first state Corolla 1. Corolla 1 states that if the $Trans$ predicate holds also $KTrans$ holds.

Corolla 1. $\forall \delta, s, \delta', s' : Trans(\delta, s, \delta', s') \rightarrow KTrans(\delta, s)$

Proof. Proof by cases over the statements of the program. Most of the cases are the defined in the same way for $Trans$ and $KTrans$ thus the corolla holds trivially for those cases. In those cases which are defined differently $Trans$ is a part of a disjunction with an additional formula in the definition of $KTrans$. Thus the corolla also holds. □

Corolla 1 can be used to state Lemma 1. The lemma shows that if a $Trans$ holds also $TransK$ and thus a transition will be made. Please note that the transitions taken by the agent must not coincide. Which is a result of the non-determinism in the transition semantic.

Lemma 1. $\forall \delta, s, \delta', s' : Trans(\delta, s, \delta', s') \rightarrow TransK(\delta, s)$

Proof. The lemma holds due to a logic consequence of Corolla 1 and the definition of $TransK$. □

Similar to the trans predicate we can state Corolla 2 for the final predicate. Corolla 2 states that if the $Final$ predicate holds also $KFinal$ holds.

Corolla 2. $\forall \delta, s : Final(\delta, s) \rightarrow KFinal(\delta, s)$

Proof. Proof by cases over the statements of the program. Most of the cases are the defined in the same way for $Final$ and $KFinal$ thus the corolla holds trivially for those cases. In those cases which are defined differently $Final$ is a part of a disjunction with an additional formula in the definition of $KFinal$. Thus the corolla also holds. □

Corolla 2 can be used to state Lemma 2. The lemma shows that if a $Final$ holds also $FinalK$ and thus a the program will terminate in the same state as before.

Lemma 2. $\forall \delta, s : Final(\delta, s, \delta', s') \leftrightarrow FinalK(\delta, s)$

Proof. The lemma holds due to a logic consequence of Corolla 2 and the definition of $FinalK$. □

Trough Lemma 1 and 2 we can conclude that our transition semantics incorporates the original transition semantics. Thus the program will neither terminate nor will it get stuck due to our changes. On the other hand our transition semantic makes it possible that the program further progress in some situations. To point out this benefit we show an illustrative example.

An agent has to pickup an object O at location L_1 bring the object to L_2 and finally bring to object to L_3. To fulfill this task the agent performs the following sequence of actions $[move(L_1), pickup(O), move(L_2), putdown(O), sensObject(O), pickup(O), move(L_3), putdown(O)]$. Now consider that the sensing action result in the conclusion that the object O is not in room L_2. There are two possibilities what could happened. The first explanation (E_1) is that the agent failed to pickup the object. The second explanation (E_2) is that the agent failed to sense the object correctly. Thus the agent would not know where the object is and could not finish its task.

Now let us consider the agent would only use E_1. Thus the agent would drive back to L_1 and try to pickup the object. Afterwards the agent would transport the object to room L_2 and sens again. Independent of the explanation this behavior would result in the state that O is in room L_2. But the agent may have spent some time to repair something which was not faulty. If the agent would only use E_2 the agent would proceed and thus would fail if the object is not in room L_2.

Now let us consider the agent would use the safe knowledge to decide how to proceed. Due to the two alternative explanations the agent does not know where the object is. The agent can not make a transition and can not decide that the task is not finished. Thus the agent gets stuck during the execution.

If the agent use the proposed transition semantics the agent can perform an active diagnosis step. This will result in the elimination of one explanation. Afterwards the agent could proceed its task.

Thus without the usage of safe knowledge the agent could either fail to finish a task or luckily finish a task but wasts time and energy through a repetition. With the help of safe knowledge the agent will not proceed a task which is not safe to assume that it is in the correct state. Without an active diagnosis step the agent could get stuck. Thus our transition semantic allows an agent working towards a goal while increasing the probability to finish its task.

5 Related Research

We start our discussion of related research with [17]. The author in [17] propose the predicate $K(s, s')$ to describe that the agent could be in s or in s'. To be more precise the agent can be in s' if she thinks she is in s. Thus we call s' to be accessible from s. Additionally the author of [17] defined $Knows(\phi, s) \doteq \forall s' : K(s, s') \rightarrow \phi(s)$ to specify if a fluent is known in every accessible world from s.

In [20] this elementary work was incorporated in the situation calculus. Furthermore a successor state axiom was defined which uses knowledge producing actions to change the relation $K(s, s')$. As a consequence within this framework it is possible that the agent starts with multiple initial situations. During the execution some of the situation are not longer accessible from s and thus "vanish" from the possible situations. Thus the agent is able to gather knowledge during its lifetime. In contrast to our work it is not possible to deal with situations where agent loses some knoweldge during its lifetime. Thus faulty actions can not be modeled.

In [21] the authors extend the initial work on knowledge and the situation calculus with a regression operator of the predicate $Knows$ and showed some theoretical properties. Within this work the authors also showed that the knowledge operator they use preserves the condition of reflexive, transitive, euclidean and symmetric specifications over the execution of actions.

Another track which incorporates knowledge into the situation calculus was proposed in [22]. Instead of the $K(s, s')$ predicate the predicate $B(s, s')$ is used with a similar meaning. The big difference

between the method proposed in [22] and [20] is that the method proposed in [22] use also a ranking over the situations. This ranking for a situation s is used to specify a plausibility over the situation depending only on the initial situation of s. The most plausible situations are used to define the belief of the agent. Thus the agent can change its knowledge over time but can not to deal with faulty actions.

One of the newest insights on knowledge in the situation calculus was proposed in [6]. The method proposed by the authors us the predicate $B(s', n, \sigma, s)$ to denote that in situation s the agent could be also in situation s' with the plausibility of n. Additionally the term σ is used to define the actions which where triggered by the agent. Thus this work is very similar to our work. There are some differences. The first difference to our work is how the ranking is calculated. The authors of [6] propose to calculate the ranking in such a way that situations which are not in contradiction with the newest sensing information have the lowest ranking. Instead our approach use a ranking which depends one the number of faulty action occurrences. Please note that the method proposed in [6] as well as in this paper could be changed in such a way that the ranking of both approaches are equal. The second main difference is that the focus of [6] was to prove certain properties of belief revision within the framework where as our paper focus on the integration of the belief management into the Indi-Golog programming execution.

The method proposed in [12] adopt the belief into the framework of the fluent calculus. The main difference between the fluent calculus and the situation calculus is that the fluent calculus is state based using the fluent which hold (or not hold) in a current situation. Instead the situation calculus use a situation term to represent the fluents. To represent the belief of an agent different states are used which are ranked with the help of a ranking function. The function also prefer the latest sensor reading and must reason about all states and there values. Thus this method differs how the ranking is calculated. Furthermore no method was shown to deal with faulty action outcomes.

Another method which uses actions and sensing for diagnosis was proposed in [1]. The method uses the sensing results to check if system components are faulty. Furthermore, actions can be performed to change the system. Thus the method is capable to deal with a dynamic system which needs to be diagnosed. To deal with multiple diagnosis actions are triggered to find a single diagnosis. Thus this work have the same general idea as the method proposed in this paper. The difference is that in this paper the diagnosis is not used to find a faulty component in a system instead to find what actions where performed in a faulty way. This makes it possible that our approach can model every diagnosis problem suitable for the approach proposed in [1]. This is achieved with the help of exegenious events which are the faulty injection events and the background knowledge describing the system. Additionally our approach can also model a faulty transition of the system. For example the switching action of a circuit went wrong. Thus the circuit is not in the expected state and behave different. This is an essential difference as a dynamic system not only could have faulty components but can also perform transitions in a faulty way.

Finally the method proposed in [13] have the same general idea of diagnosis and sensing. The method diagnose a printing system through its outcome and use a planing unit to use different components of the printer for the next jobs to find a small set of possible faulty components. Beside the goal to find the faulty components the method also try to fulfill the current production goals. Thus the method makes it possible that the impact on the production through a faulty component is as small as possible. The main difference to the method proposed in this paper is the focus on diagnosis a faulty system instead of diagnosing a faulty action outcome. Furthermore the sensing action can always been performed which is in contrast to our approach which have to deal with a changing world which may prevent the agent from performing a sensing action.

6 Conclusion and Future work

In this paper we briefly discuss how the epistemic program execution of the well-known IndiGolog interpreter can be improved by using active diagnosis for situations. The main idea is to not abort the program execution once the needed knowledge is not secured. If a state is reached where neither transition nor termination can be safely decided an active diagnosis step is performed to increase the secured knowledge available for program execution. Moreover, a connection between the semantics of the possible situations obtained by diagnosis and the big corpus of research on epistemic approaches was made.

In particular the latter connection seems promising to investigate in more detail. Besides, problems with the runtime in particular of the active diagnosis step (pruning of situations) there are also other open issues. In particular there are issues in the semantics of the changed program transition. For instance should program transition be given a chance once the agent is not able to decide termination?

REFERENCES

[1] Chitta Baral, Sheila McIlraith, and Tran Cao Son, 'Formulating diagnostic problem solving using an action language with narratives and sensing', in *KR*, pp. 311–322, (2000).

[2] Giuseppe De Giacomo, Yves Lespérance, and Hector J Levesque, 'Congolog, a concurrent programming language based on the situation calculus', *Artificial Intelligence*, **121**(1), 109–169, (2000).

[3] Giuseppe De Giacomo, Yves Lespérance, Hector J Levesque, and Sebastian Sardina, 'Indigolog: A high-level programming language for embedded reasoning agents', in *Multi-Agent Programming:*, 31–72, Springer, (2009).

[4] Giuseppe De Giacomo and Hector J Levesque, 'An incremental interpreter for high-level programs with sensing', in *Logical Foundations for Cognitive Agents*, 86–102, Springer, (1999).

[5] Johan De Kleer and Brian C Williams, 'Diagnosing multiple faults', *Artificial intelligence*, **32**(1), 97–130, (1987).

[6] James P Delgrande and Hector J Levesque, 'Belief revision with sensing and fallible actions.', in *KR*, (2012).

[7] Robert Demolombe and Maria Pilar Pozos Parra, 'A simple and tractable extension of situation calculus to epistemic logic', in *Foundations of Intelligent Systems*, eds., ZbigniewW. Ra and Setsuo Ohsuga, volume 1932 of *Lecture Notes in Computer Science*, 515–524, Springer Berlin Heidelberg, (2000).

[8] Thomas Eiter, Michael Fink, and Ján Senko, 'Kmonitor–a tool for monitoring plan execution in action theories', in *Logic Programming and Nonmonotonic Reasoning*, 416–421, Springer, (2005).

[9] 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, 31–72, Springer, 2009.

[10] Stephan Gspandl, Ingo Pill, Michael Reip, Gerald Steinbauer, and Alexander Ferrein, 'Belief Management for High-Level Robot Programs', in *The 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 900–905, Barcelona, Spain, (2011).

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

[12] Yi Jin and Michael Thielscher, 'Representing beliefs in the fluent calculus', in *ECAI*, volume 16, p. 823, (2004).

[13] Lukas Kuhn, Bob Price, Johan de Kleer, Minh Do, and Rong Zhou, 'Pervasive diagnosis: Integration of active diagnosis into production plans', in *proceedings of AAAI*, (2008).

[14] Hector J. Levesque, 'Planning with loops', in *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pp. 509–515, San Francisco, CA, USA, (2005). Morgan Kaufmann Publishers Inc.
[15] Hector J Levesque, Raymond Reiter, Yves Lesperance, Fangzhen Lin, and Richard B Scherl, 'Golog: A logic programming language for dynamic domains', *The Journal of Logic Programming*, **31**(1), 59–83, (1997).
[16] J. McCarthy, 'Situations, Actions and Causal Laws', Technical report, Stanford University, (1963).
[17] Robert C Moore, 'A formal theory of knowledge and action', Technical report, DTIC Document, (1984).
[18] Bernhard Nebel, Christian Dornhege, and Andreas Hertle, 'How much does a household robot need to know in order to tidy up?', in *Proceedings of the AAAI Workshop on Intelligent Robotic Systems, Bellevue, WA*, (2013).
[19] R. Reiter, *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*, MIT Press, 2001.
[20] Richard B Scherl and Hector J Levesque, 'The frame problem and knowledge-producing actions', in *AAAI*, volume 93, pp. 689–695. Citeseer, (1993).
[21] Richard B Scherl and Hector J Levesque, 'Knowledge, action, and the frame problem', *Artificial Intelligence*, **144**(1), 1–39, (2003).
[22] Steven Shapiro, Maurice Pagnucco, Yves Lespérance, and Hector J Levesque, 'Iterated belief change in the situation calculus', in *KR*, pp. 527–538, (2000).