# Diagnostic Reasoning for Robotics using Action Languages

Esra Erdem[1], Volkan Patoglu[1], and Zeynep Gozen Saribatur[2]*

[1] Sabanci University, Istanbul, Turkey
[2] Vienna University of Technology, Vienna, Austria

**Abstract.** We introduce a novel diagnostic reasoning method for robotic systems with multiple robots, to find the causes of observed discrepancies relevant for plan execution. Our method proposes (i) a systematic modification of the robotic action domain description by utilizing defaults, and (ii) algorithms to compute a smallest set of diagnoses (e.g., broken robots) by means of hypothetical reasoning over the modified formalism. The proposed method is applied over various robotic scenarios in cognitive factories.

**Keywords:** diagnostic reasoning, action languages, answer set programming

## 1 Introduction

For the fault-awareness and reliability of robotic systems (e.g., in cognitive factories or service robotics), it is essential that the robots have cognitive skills, such as planning their own actions, identifying discrepancies between the expected states and the observed states during plan execution, checking whether these discrepancies would lead to a plan failure, diagnosing possible causes of relevant discrepancies, learning from earlier diagnoses, and finding new plans to reach their goals. In this paper, we focus on diagnostic reasoning for robotics.

Consider, for instance, a cognitive factory [32, 11]. In a typical cognitive factory, each workspace is medium sized with 3–12 heterogeneous robots; there may be many workspaces, each focusing on a different task. On the other hand, each of these robots has many components, failure of which may cause abnormalities in the manufacturing process. Factory shut-downs for diagnosis and repairs are very costly; hence, it is required that diagnosis is performed accurately and fast. Furthermore, it is essential that the monitoring agent(s) have the capability of identifying further details on diagnosis (broken robot components, actions that could not be executed due to broken robots/components), and learning from earlier diagnoses and failures.

With these motivations, we introduce a diagnostic reasoning method to find a smallest set of broken robots (and their components) that cause the relevant discrepancies observed during plan execution. Our method starts with the robotic action domain description used to compute a plan, the part of the plan executed from the initial state until the current state, and a set of observations about the current state. The robotic domain description is represented in an action description language [16], like $\mathcal{C}+$ [18]. Then,

* Z. G. Saribatur's work was carried out during her graduate studies at Sabancı University.

our method (i) applies a systematic modification of the robotic action domain description by utilizing defaults, and (ii) computes a smallest set of diagnoses by means of hypothetical reasoning over the modified formalism.

The modification of the domain description is essential to be able to generate possible causes of the observed discrepancies that would lead to plan failures. The use of defaults help generation of possible causes for discrepancies (such as broken robots), as well as generation of further information (such as which actions are prevented from execution due to these causes). We have proven that our proposed modification conservatively extends the planning domain description, which is important for generating such further information.

The computation of smallest sets of diagnoses is done by means of "diagnostic queries" expressed in an action query language, like $\mathcal{Q}$ [16], and by using efficient automated reasoners, such as SAT solvers and ASP solvers. Two algorithms are proposed for this computation: one of them relies more on hypothetical reasoning and can be used in conjunction with both sorts of reasoners; the other algorithm relies on aggregates and and optimization statements of answer set programming (ASP) [24, 27, 22, 23, 3].

Both algorithms synergistically integrate diagnostic reasoning with learning from earlier experiences and probabilistic geometric reasoning. As demonstrated by our experiments, learning improves the computational efficiency and quality of diagnoses, since it allows robots to utilize their previous experiences (e.g., which robots or robot components are more probable to be broken). As shown by examples, geometric reasoning improves accuracy of diagnoses by considering feasibility of robotic actions.

## 2 Preliminaries

We model dynamic domains like cognitive factories as transition diagrams – directed graphs where the nodes characterize world states and the edges represent transitions between states caused by (non)occurrences of actions. We represent transition diagrams formally in the nonmonotonic logic-based action description language $\mathcal{C}+$ [18].

Various sorts of reasoning tasks can be performed over transition diagrams, such as planning and prediction. We describe reasoning tasks by means of formulas in an action query language $\mathcal{Q}$ [16], and utilize SAT/ASP solvers to compute a solution.

This logic-based reasoning framework allows integration of low-level feasibility checks (e.g., collision checks for robots) performed externally by the state-of-the-art geometric reasoners. In this section, we will briefly describe these preliminaries over a cognitive toy factory scenario.

### 2.1 A cognitive toy factory

As a running example, we consider the cognitive toy factory workspace of [12], where a team of multiple robots collectively works toward completion of an assigned manufacturing task. In particular, the team manufactures nutcracker toy soldiers through the sequential stages of cutting, carving and assembling. The workspace contains an assembly line to move the toys and a pit stop area where the worker robots can change their end-effectors. It also includes static obstacles.

The team is heterogeneous, composed of two types of robots with different capabilities. Worker robots operate on toys, they can configure themselves for different stages of processes; charger robots maintain the batteries of workers and monitor team's plan. All robots can move from any grid cell to another one following straight paths.

## 2.2 Representation of a robotic domain

The signature of an action domain description in $\mathcal{C}+$ consists of two sorts of multi-valued propositional constants, fluent constants $\mathcal{F}$ and action constants $\mathcal{A}$, along with a nonempty set $Dom(c)$ of possible values for each constant $c$. Atoms are of the form $c = v$ where $c$ is a constant and $v$ is a value in $Dom(c)$. If $c$ is a boolean constant then we adopt the notations $c$ and $\neg c$. We assume that action constants are boolean.

Transition diagrams modeling dynamic domains can be described by "causal laws" over such a signature. For instance, the following causal law describes a direct effect of a robot $r$ moving in the *Right* direction by $u$ units from a location $x$ on the x-axis: the location of the robot becomes $x + u$ at the next state after the execution of this action.

$$move(r, Right, u) \textbf{ causes } xpos(r) = x + u \textbf{ if } xpos(r) = x. \tag{1}$$

The causal law below describes a precondition of this action: a worker robot $w$ cannot move $u$ units in any direction $d$ since its battery lasts for only $bl<u$ units of movement:

$$\textbf{nonexecutable } move(w, d, u) \textbf{ if } battery(w) = bl \quad (bl < u). \tag{2}$$

Similarly, causal laws can represent [18] ramifications of actions, noconcurrency constraints, state/transition constraints, the commonsense law of inertia.

We understand an action domain description as a finite set of definite causal laws.

## 2.3 Reasoning about a robotic domain

A *query* in $\mathcal{Q}$ [16] is a propositional combination of atomic queries of the two forms, $F$ **holds at** $t$ or $A$ **occurs at** $t$, where $F$ is a fluent formula, $A$ is an action formula, and $t$ is a time step.

The meaning of a query is defined in terms of histories. A *history* of an action domain description $\mathcal{D}$ is an alternating sequence $s_0, A_0, s_1, \ldots, s_{n-1}, A_{n-1}, s_n$ $(n \geq 0)$ of states and actions, denoting a path in the transition diagram described by $\mathcal{D}$. States (resp. actions) can be considered as functions from fluent constants (resp. action constants) to their relevant domains of values. Then each state (resp. action) can be denoted by a set of fluent (resp. action) atoms.

A query $Q$ of the form $F$ **holds at** $t$ (resp. $A$ **occurs at** $t$) is *satisfied* by such a history if $s_t$ satisfies $F$ (resp. if $A_t$ satisfies $A$). For non-atomic queries, satisfaction is defined by truth tables of propositional logic. A query $Q$ is *satisfied* by an action description $\mathcal{D}$, if there is a history of $\mathcal{D}$ that satisfies $Q$.

Let $F$ and $G$ be fluent formulas representing an initial state and goal conditions. We can describe the problem of finding a plan of length $k$, with a query of the form $F$ **holds at** $0 \wedge G$ **holds at** k. Similarly, we can describe the problem of predicting the resulting state after an execution of an action sequence $A_0, \ldots, A_{n-1}$ at a state described by a fluent formula $F$, with a query of the form

$$F \textbf{ holds at } 0 \wedge \wedge_i A_i \textbf{ occurs at } \text{i}. \tag{3}$$

It is shown [18] that an action description in $\mathcal{C}+$ can be transformed into a propositional theory and into an ASP program. Based on these sound and complete transformations, the software systems CCALC [25, 18] and CPLUS2ASP [5] turn an action domain and a given query into the input languages of a SAT solver and an ASP solver.

### 2.4 Hybrid robotic planning

Geometric reasoning, such as motion planning and collision checks, can be integrated into an action description in $\mathcal{C}+$ by means of "external atoms" [9] (in the spirit of semantic attachments in theorem proving [31]). The idea is to compute the truth values of external atoms externally, and utilize these results, as needed, while computing plans.

For instance, consider an external atom $\&\mathit{collision}[r, x1, y1, x2, y2]$ that returns true if a robot $r$ collides with some static obstacles while moving from $(x1, y1)$ to $(x2, y2)$. This atom can be evaluated by a program implemented in C++ utilizing a motion planner. With this atom, we can describe a precondition of a diagonal move action ("the robot can move diagonally if it does not collide with any static obstacles"):

$$\begin{array}{l}\textbf{nonexecutable } \mathit{move}(w, \mathit{Right}, \mathit{u1}) \wedge \mathit{move}(w, \mathit{Up}, \mathit{u2}) \\ \quad \textbf{if } \mathit{xpos}(w) = \mathit{x1} \wedge \mathit{ypos}(w) = \mathit{y1} \textbf{ where } \&\mathit{collision}[w, \mathit{x1}, \mathit{y1}, \mathit{x1}{+}\mathit{u1}, \mathit{y1}{+}\mathit{u2}].\end{array}$$

Further explanations and examples of the use of external atoms in robotic action domain descriptions, and a systematic analysis of various forms of integration of feasibility checks with planning can be found in [4, 10, 13].

## 3 Diagnostic Reasoning

To predict failures as early as possible and to recover from failures, values of some fluents can be monitored by an agent. If during the execution of a plan, a discrepancy is detected between the observed values of monitored fluents and the expected values of monitored fluents, then the monitoring agent can check whether the detected discrepancy might lead to any failures during the execution of the rest of the plan. If the discrepancy is relevant for the rest of the plan, then the monitoring agent has to make some decisions (e.g., replanning) to reach the goals. We propose the use of diagnostic reasoning to identify the cause of a discrepancy (e.g., robots may be broken), and then find a relevant recovery with the possibility of repairs.

### 3.1 Discrepancies

Let $\mathcal{D}$ be a domain description. Let $P = \langle A_0, A_1, \ldots, A_{n-1} \rangle$ be a plan that is being executed from an initial state $s_0$. We denote by $P_t$ the sequence $A_0, A_1, \ldots, A_{t-1}$ of actions in the plan $P$ executed until step $t$. Let $o_t$ be an observed state of the world at step $t$, where the observed values of fluents can be obtained by sensors. The expected state $e_t$ of the world at step $t$ can be computed by a prediction query of the form (3) with the initial state $s_0$ and the sequence $P_t$ of actions.

We say that there is a *discrepancy* at step $t$ if the observed state and the expected state are different, $o_t \neq e_t$. A discrepancy is *relevant* to the rest of the plan if the rest of the plan can not be executed from the observed state or does not reach a goal state. This definition coincides with weakly $k$-irrelevant discrepancies when $k = 1$ [8] and with the definition of relevancy as in [17].

## 3.2 Diagnosis: Identifying broken robots

In a cognitive factory setting, a diagnosis for a discrepancy can be identified by a set of broken robots. To be able to find such a diagnosis, we first modify the action domain description to be able to perform diagnostic reasoning. Then, we define diagnosis and diagnostic queries to compute diagnoses over the modified domain description.

**Modifying the domain description for diagnosis** Let $R$ denote the set of robots in a cognitive factory, that may get broken. Let *disables* : $2^R \times \mathcal{A} \times \mathcal{F}$ be a relation to describe which actions are affected and how, if a set of robots were broken: *disables*$(X, a, F)$ expresses that if the set $X$ of robots is broken, then the effect of action $a \in \mathcal{A}$ performed by some robots in $X$ on a fluent $F$ is disabled. Note that *disables* can be obtained automatically from the causal laws that describe effects of actions.

To find a diagnosis for the observed relevant discrepancies, we modify the domain description $\mathcal{D}$ in three stages as follows, and obtain a new domain description $\mathcal{D}_b$.
Step 1: To indicate whether a robot $r \in R$ is broken or not, we introduce a simple fluent constant of the form *broken*$(r)$.

By default, the robots in $R$ are assumed to be not broken. We express this for every robot $r \in R$, with a causal law:

$$\textbf{default } \neg broken(r). \tag{4}$$

Meanwhile, every robot $r$ may get broken at any time:

$$\textbf{caused } broken(r) \textbf{ if } broken(r) \textbf{ after } \neg broken(r) \tag{5}$$

and if a robot $r$ becomes broken it remains broken:

$$\textbf{caused } broken(r) \textbf{ after } broken(r). \tag{6}$$

For every $r \in R$, we add the causal laws (4)–(6) to the domain description $\mathcal{D}$.

It is important to emphasize the usefulness of the nonmonotonicity of $\mathcal{C}+$, which allows us to represent defaults (4) as well as nondeterminism (5), and thus does not necessitate introduction of non-robotic actions like *break*.
Step 2: If a robot is broken, then it may affect preconditions and effects of the actions in the executed plan. It is good to know which actions could not be executed due to which sort of broken robots, so that we can learn/infer some new knowledge that might be used later for more accurate diagnoses and more effective repairs.

For that, for each (concurrent) action $A = \{a_0, ..., a_n\}$, we introduce a simple boolean fluent constant $pre(\{a_0, ..., a_n\})$ to express whether or not its preconditions hold; brackets are dropped when $|A| = 1$. The values of these fluents are considered as true by default; so we add to $\mathcal{D}$ the causal laws:

$$\textbf{default}\ pre(\{a_0, ..., a_n\}). \tag{7}$$

We then describe under which conditions an action's preconditions are violated, by replacing every causal law

$$\textbf{nonexecutable} \bigwedge_{a_i \in A} a_i\ \textbf{if}\ G$$

in $\mathcal{D}$, where $G$ is a fluent formula, with the causal law

$$\textbf{caused}\ \neg pre(\{a_0, ..., a_n\})\ \textbf{if}\ G. \tag{8}$$

For instance, the causal laws (2) are replaced by the causal laws:

$$\textbf{caused}\ \neg pre(move(w, d, u))\ \textbf{if}\ battery(w) = bl\ \ (bl < u).$$

We then modify the causal laws describing the effects of actions, to express that they can be observed under the additional condition that the preconditions of these actions are satisfied. For that, we replace every causal law of the form

$$a\ \textbf{causes}\ F\ \textbf{if}\ G \tag{9}$$

in $\mathcal{D}$, where $F$ and $G$ are fluent formulas, with the dynamic causal law

$$a\ \textbf{causes}\ F\ \textbf{if}\ G \wedge pre(a). \tag{10}$$

For instance, the causal laws (1) describing some effects of the move action are replaced by the following:

$$move(r, Right, u)\ \textbf{causes}\ xpos(r) = x + u\ \textbf{if}\ xpos(r) = x \wedge pre(move(r, Right, u)).$$

Step 3: For every action $a$, let $a_R$ denote the robots in $R$ that take part in the execution of $a$. We reflect the influence of broken robots on direct effects of actions by replacing every causal law (9) in $\mathcal{D}$ such that *disables*$(a_R, a, F)$ holds, with the causal law

$$a\ \textbf{causes}\ F\ \textbf{if}\ G \wedge \bigwedge_{r \in a_R} \neg broken(r), \tag{11}$$

which expresses that the direct effects of actions are observed as expected only if the relevant robots are not broken.

In our cognitive factory scenario, a charger robot cannot dock to a worker robot $w$ if the worker robot is broken, i.e., *disables*$(\{w\}, attach(w), attached(w))$ (the direct effect of $attach(w)$ action on the fluent $attached(w)$ is not observed if the worker robot $w$ is broken). Then, the causal law

$$attach(w)\ \textbf{causes}\ attached(w)\ \textbf{if}\ pre(attach(w))$$

obtained after Step 2 is replaced by the causal law

$$attach(w)\ \textbf{causes}\ attached(w)\ \textbf{if}\ pre(attach(w)) \wedge \neg broken(w).$$

The following proposition shows that if a query is satisfied by $\mathcal{D}$ then it is also satisfied by $\mathcal{D}_b$; but not vice-versa. In that sense, $\mathcal{D}_b$ "conservatively extends" $\mathcal{D}$. This is particularly important to be able to reason about executions of a plan (earlier computed with $\mathcal{D}$) with respect to $\mathcal{D}_b$ to find diagnoses.

**Proposition 1** *Every query satisfied by $\mathcal{D}$ is satisfied by $\mathcal{D}_b$.*

The proof of the proposition requires the following lemmas. Suppose that $\mathcal{D}$ is defined over a set $\mathcal{F}$ of fluent constants and a set $\mathcal{A}$ of boolean action constants. Let $\mathcal{M}$ be the set of all fluent constants of the forms $\mathsf{broken}(r)$ and $pre(A)$ that are introduced to the signature of $\mathcal{D}$, to transform $\mathcal{D}$ into $\mathcal{D}_b$. Then, every state $s'$ of $\mathcal{D}_b$ is a function mapping every constant $c$ in $\mathcal{F} \cup \mathcal{M}$ to a value in $Dom(c)$.

**Lemma 1** *For every state $s$ of $\mathcal{D}$, there are states $s_b$ of $\mathcal{D}_b$ such that (i) $s_b|_{\mathcal{F}} = s$, (ii) $s_b$ maps every fluent constant $\mathsf{broken}(r)$ to $\mathsf{false}$, and (iii) for every causal law (8), $s_b$ satisfies $G \supset \neg pre(\{a_0, \ldots, a_n\})$.*

**Lemma 2** *For every transition $\langle s, A, s'\rangle$ of $\mathcal{D}$, there are transitions $\langle s_b, A, s'_b\rangle$ of $\mathcal{D}_b$ such that (i) $s_b|_{\mathcal{F}} = s$, (ii) $s'_b|_{\mathcal{F}} = s'$, (iii) both $s_b$ and $s'_b$ map every fluent constant $\mathsf{broken}(r)$ to $\mathsf{false}$, and (iv) $s_b$ maps every fluent constant $pre(Z)$ ($Z \subseteq A$) to $\mathsf{true}$.*

**Diagnosis** Let us characterize a state $s$ by the conjunction $F_s$ of atoms in $s$; and an action $A$ by the conjunction $E_A$ of atoms in $A$.

Intuitively, a diagnosis for a discrepancy detected at time step $t$ of a plan $P$ is a set of robots that, when broken, provides a possible execution of the plan from the initial state to the observed state (i.e., they do not lead to an "inconsistency"). Formally:

**Definition 1** *A* diagnosis problem, *$DP$, is characterized by a tuple $\langle \mathcal{D}_b, R, s_0, P_t, o_t\rangle$ where $t$ is the time step when a discrepancy is detected, $P_t = \langle A_0, \ldots, A_{t-1}\rangle$ is the sequence of actions assumed to be executed at an initial state $s_0$ until time step $t$, and $o_t$ is the observed state at time step $t$. A* solution *of $DP$ is a set $X \subseteq R$ of broken robots such that $\mathcal{D}_b$ satisfies the query*

$$\begin{array}{l} F_{s_0}\ \textbf{\textit{holds at}}\ 0\ \wedge \bigwedge_{A_i \in P_t} E_{A_i} \textbf{\textit{occurs at}}\ i\ \wedge F_{o_t} \textbf{\textit{holds at}}\ t\ \wedge \\ \bigwedge_{r \in X} broken(r)\ \textbf{\textit{holds at}}\ t \wedge \bigwedge_{r \in R-X} \neg broken(r)\ \textbf{\textit{holds at}}\ t. \end{array} \qquad (12)$$

*We also say that $X$ is a* diagnosis *of the discrepancy detected at time $t$.*

Note that further information about which actions' preconditions are violated can be obtained from the history of $\mathcal{D}_b$ that satisfies the query, since atoms of the form $\neg pre(A)$ are part of the state information.

The query (12) checks if the observed state $o_t$ can be reached from the initial state $s_0$ by only executing the plan $P_t$, if the robots in $X$ were broken. Note that, by the definition of a discrepancy, we know that (12) will not be satisfied by $\mathcal{D}_b$ if $X = \emptyset$, since the observed state is different from the state that is expected to be reached from $s_0$ by $P_t$.

Generally, there will be more than one diagnosis for a discrepancy according to the definition above. On the other hand, in practice, a discrepancy is caused by a small set of broken robots. Therefore, it is reasonable to find as few broken instances as possible to explain the discrepancy.

**Definition 2** *A diagnosis $X$ is a* minimum-cardinality *diagnosis if there does not exist any other diagnosis $X'$ such that $|X'| < |X|$.*

### 3.3 Finding a minimum-cardinality diagnosis

We introduce two algorithms to find minimum-cardinality diagnoses by means of diagnostic queries as described above, using SAT solvers or ASP solvers.

In the first algorithm, to find a minimum-cardinality diagnosis for a diagnosis problem $DP = \langle \mathcal{D}_b, R, s_0, P_t, o_t^M \rangle$, for each $X \subseteq R$ with increasing cardinalities, we iteratively check whether $X$ is a solution for $DP$, i.e., whether $\mathcal{D}_b$ satisfies (12). We can utilize this algorithm with both SAT solvers and ASP solvers, thanks to the sound and complete transformations and software systems mentioned in the preliminaries.

The second algorithm is applicable with ASP solvers only, thanks to the possibility of representing aggregates and optimization statements. For instance, the following optimization statement minimizes the total number of broken robots:

$$\#\mathit{minimize}\ [\mathit{broken}(r) : \mathit{robot}(r)]. \tag{13}$$

## 4 Geometric Reasoning for Diagnosis

Embedding geometric reasoning in the domain description $\mathcal{D}$ (and thus $\mathcal{D}_b$) by means of external atoms, allows us to integrate diagnostic reasoning with geometric reasoning. Let us show the importance of such an integration over an example, as in [12]. Consider the execution of some part of a plan from a given state as in Figure 2, and the observed state of the world at step 3 as in Figure 1. There is a discrepancy at step 3. To find a diagnosis for this discrepancy, we ask diagnostic queries (12). To check whether the charger is broken, we ask “is it possible to reach from the initial state to the observed state by executing the plan, if the charger were broken?”

Without consideration of geometric reasoning, the answer to this question is “no” since the goal is not reachable when the plan is executed as shown in Figure 3. However, such an execution is not feasible in the real world because the worker robot cannot move diagonally over the charger robot at Step 1 (due to collisions). On the other hand, with consideration of geometric reasoning, the answer to the diagnostic query is “yes” since the goal is reachable when the plan is executed as shown in Figure 4. Note that, due to collisions, the preconditions of the action of the worker robot moving diagonally is not possible, i.e., $\neg\mathit{pre}(\{move(w, Up, 2), move(w, Right, 2)\})$ holds. Therefore, the detected discrepancy is correctly diagnosed with hybrid diagnostic reasoning and false negatives are avoided.

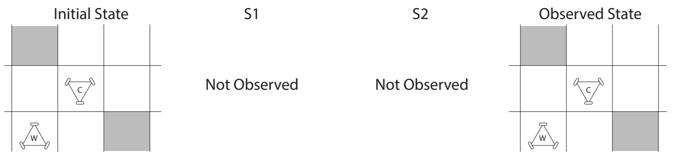

**Fig. 1.** Observed state of the world at step 3.

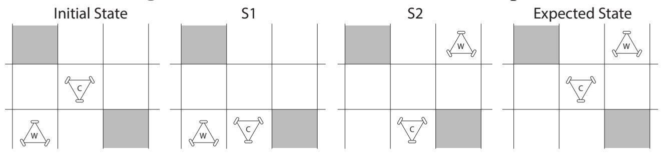

**Fig. 2.** Expected execution of a plan until step 3.

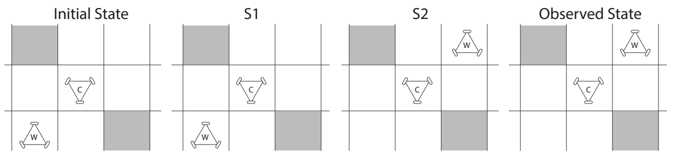

**Fig. 3.** Diagnostic reasoning without geometric reasoning

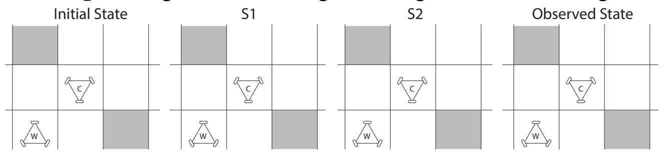

**Fig. 4.** Diagnostic reasoning with geometric reasoning

## 5 Extensions of Our Method

**Further diagnoses: broken robotic parts.** Usually robots are broken if some of their components are broken. Moreover, each broken component may play a role on the preconditions/effects of different actions. To extend diagnostic reasoning to robotic components, we modify the first and third steps of our procedure to obtain $\mathcal{D}_b$ from $\mathcal{D}$, as well as the algorithms to compute smallest diagnoses. The definition of a diagnosis remains the same.

For these modifications, we define $R_p$ as the set of pairs $(r, i)$ where $r \in R$ and $i$ is a component of $r$ that may be broken. For instance, for a cognitive toy factory scenario with one worker robot $w$ and with one charger robot $c$, $R_p = \{(w, Arm), (w, Inlet), (c, Plug), (w, Base), (c, Base)\}$. We extend the *disables* relation: $\textit{disables}_p : 2^{R_p} \times \mathcal{A} \times \mathcal{F}$. For instance, $\textit{disables}_p(\{(w, Inlet)\}, charge(c), battery(w))$ expresses that, if the inlet of the worker robot is broken then the effect of charging will not be observed as expected on its battery level. In the representation $\mathcal{D}_b$, (i) we introduce new fluent constants of the form $\textit{broken}_i(r)$ for every $(r, i) \in R_p$ and add relevant causal laws for them, and (ii) we modify the causal laws (11) according to $\textit{disables}_p$. The algorithms are modified to ensure that the computed smallest diagnosis contains minimum number of broken components. Details are omitted for space restrictions.

**Learning from previous diagnoses.** If a discrepancy detected in a plan execution is associated with several robots, there may be more than one potential diagnosis for the discrepancy. If the computed diagnosis cannot be verified (by an agent) as correct, then a different diagnosis can be computed by modifying the diagnostic query with a constraint. This process goes on until a correct diagnosis. As in [12], we reduce the number of such uninformed iterations due to incorrect diagnoses, by utilizing learning from earlier correct diagnoses of the discrepancies. The idea is to maintain and update information about the likelihood of robots'/components' failures according to computed correct diagnoses; and if there is a robot/component which has a history of being broken more often, then to consider it as part of a potential diagnosis before the others.

To describe how often a robot $r$ (resp. a component $i$ of a robot $r$) is diagnosed correctly as broken, we introduce atoms of the form $\mathit{weight}(r, w)$ (resp. $\mathit{weight}(r, i, w)$ ) where $w$ is a number, which can store the number of times that $r$ (resp. $i$ of $r$) was correctly diagnosed to be broken, or the probability of $r$ (resp. $i$ of $r$) being diagnosed correctly so far. To find the most probable diagnosis, in the first algorithm proposed for finding a smallest diagnosis, we consider subsets of robots/components with larger weights before the others. In the second algorithm, we add a new optimization statement to the ASP program, which tries to maximize the total weight of broken components. This statement is added after the minimization statement (13), to maximize the total weight of broken components among the minimum-cardinality ones.

Extending the experimental evaluations of [12], we performed several experiments over various cognitive toy factory scenarios to show the usefulness of learning in the first algorithm for finding a minimum-cardinality diagnosis by using the SAT solver MINISAT (Version 2.0) [7], and in the second algorithm by using the ASP solver CLASP (Version 2.1.3) [14]. Experiments are based on dynamic simulations, where kinematic and geometric constraints of robots are considered. For geometric reasoning, we use probabilistic motion planners and collision checkers available in open-source frameworks, like OPENRAVE [6]. In each scenario, we first performed the experiments without learning. Based on the computed diagnoses, we assigned the weights for robots/components, and then performed the experiments with learning. For each instance, the CPU time in seconds, and the number of iterations to compute the correct diagnosis with correct set of broken components are illustrated in Table 1 (which extends the results in Table 1 of [12]). The results are obtained on a Linux server with 16 Intel E5-2665 CPU cores (2.4GHz) and 64GB memory.

As in [12], we observe 1) in both algorithms, the number of iterations to find a correct diagnosis significantly decreases as learning is utilized; 2) as the size of the team and the cardinality of the diagnosis increase, the computation time to find a correct diagnosis increases. In addition to the observations of [12]: 3) The number of iterations in ASP is less than the number of iterations with SAT solver, since the optimization statement of ASP is effective in decreasing number of iterations to find a correct diagnosis.

**Behavioral modes.** A further extension of our approach can be by behavioral modes [20]. Since our approach is based on representing actions and change in the environment (unlike the related work), behaviors of the robotic system are already modeled. We suppose each robot has three modes of behavior: normal mode (functions as expected), broken mode (does not function at all), or unknown mode.The broken modes are depicted by *disables* relation; the behaviors are described by causal laws. In the normal mode, the behavior (i.e., expected effects of relevant actions) is described by laws like (1). In the broken mode, the behavior (i.e., by default nothing changes) is described by the com-

**Table 1.** Experimental Evaluation of Learning

| Scenario | Number of iterations and CPU time [secs] | | | |
|---|---|---|---|---|
| | ASP | | SAT | |
| | w/o learning | w/ learning | w/o learning | w/ learning |
| 1 charger, 1 wet, 2 dry<br>Discrepancy at Step 12, Diagnosis cardinality=1 | 2<br>6.51 secs | 1<br>3.21 secs | 9<br>52.24 secs | 4<br>26.49 secs |
| 1 charger, 1 wet, 2 dry<br>Discrepancy at Step 8, Diagnosis cardinality=2 | 2<br>4.14 secs | 1<br>2.06 secs | 21<br>67.43 secs | 11<br>39.85 secs |
| 1 charger, 1 wet, 2 dry<br>Discrepancy at Step 10, Diagnosis cardinality=3 | 2<br>4.28 secs | 1<br>2.14 secs | 25<br>85.88 secs | 23<br>84.16 secs |
| 1 charger, 2 wet, 2 dry<br>Discrepancy at Step 8, Diagnosis cardinality=1 | 2<br>7.66 secs | 1<br>3.85 secs | 7<br>83.70 secs | 4<br>42.97 secs |
| 1 charger, 2 wet, 2 dry<br>Discrepancy at Step 12, Diagnosis cardinality=2 | 2<br>9.10 secs | 1<br>4.56 secs | 24<br>142.18 secs | 12<br>80.35 secs |
| 1 charger, 2 wet, 2 dry<br>Discrepancy at Step 13, Diagnosis cardinality=3 | 2<br>9.86 secs | 1<br>4.93 secs | 39<br>197.01 secs | 25<br>127.25 secs |
| 2 charger, 2 wet, 2 dry<br>Discrepancy at Step 6, Diagnosis cardinality=1 | 2<br>6.27 secs | 1<br>3.13 secs | 8<br>97.00 secs | 4<br>49.30 secs |
| 2 charger, 2 wet, 2 dry<br>Discrepancy at Step 14, Diagnosis cardinality=2 | 2<br>13.29 secs | 1<br>6.63 secs | 31<br>195.17 secs | 13<br>102.48 secs |
| 2 charger, 2 wet, 2 dry<br>Discrepancy at Step 12, Diagnosis cardinality=3 | 2<br>11.57 secs | 1<br>5.78 secs | 49<br>240.90 secs | 30<br>154.85 secs |

monsense law of inertia. In the unknown mode, we assume by default nothing changes. Considering failures of robotic components leads to more number of behavioral modes. The broken modes are specified by $\mathit{disables}_p$ relation, and behaviors are described by causal laws. Therefore, associating diagnoses explicitly with such behavioral modes may be possible; it is left as a future work.

## 6 Related Work

Our work on diagnosis is similar to conflict-based model-based diagnosis [28, 21]: the diagnostic query checks whether broken robots would lead to an inconsistency with respect to the observations or not. It is also different in several ways. First, we consider dynamic domains with actions and change, rather than a static system like circuits. Second, our logical framework is nonmonotonic and thus the definition of a diagnosis is different from the existing definitions. Third, motivated by robotics applications, our approach to diagnostic reasoning is integrated with geometric reasoning and learning.

Later, the model-based diagnosis approach of [28, 21] is extended to dynamic domains, using the logic-based action languages, such as situation calculus [26, 19] and fluent calculus [30], and utilizing planning [29]. Our work is different from these approaches not only because of differences between the underlying formalisms but also due to diagnosis definition (faulty actions vs. components), assumptions (faulty components/actions may indirectly prevent execution of the rest of the plan, or not; our method is applicable in either case), methods (introducing abnormality predicates and utilizing nonmonotonicity by minimizing its extension, vs. expressing defaults; introducing further transformations to domain description beyond abnormality predicates).

A more closely related line of research is diagnostic reasoning in answer set programming and action languages [8, 1, 2, 11, 33, 15] due to the common underlying formalisms. Our approach is different from these works in several ways. Eiter et al. [8] define a diagnosis as a “point of failure” which describes at which state and when the plan diverges from its expected evolution; we consider a diagnosis as a set of broken robots/components. Thanks to the modified domain description $\mathcal{D}$, we can identify which action has failed and when, as part of an answer to the diagnostic query. Balduccini and Gelfond [1], Baral et al. [2], and Gelfond and Kahl [15] (like [26]) define a diagnosis as a set of broken components; they introduce a “break” action to define what might be “broken”. In $\mathcal{C}+$, no such non-robotic action is necessary; the causal laws (5) are sufficient. Gelfond and Kahl also suggest using consistency-restoring rules [1] to find minimal diagnoses. Erdem et al. [11] define a diagnosis as a set of broken robots, but their method does not generalize to scenarios where broken robots prevent the execution of an action in the rest of the plan and does not provide further information about broken components or failed actions. Zhang et al. [33] generate explanations for discrepancies in terms of exceptions to the defaults that hold at the initial state.

Our approach to diagnostic reasoning can be extended with repair planning by introducing “repair” actions into the robotic domain description $\mathcal{D}$ as in [1, 2]. Such repairs can reduce the number of replanning needed to recover from plan failures. Extending our method with repairs is not discussed due to page limit.

## 7 Conclusion

We have introduced a diagnostic reasoning method which utilizes expressive formalisms of action languages and answer set programming, and efficient automated reasoners (SAT solvers and ASP solvers). This method integrates geometric reasoning (for feasibility checks of robotic actions), to eliminate false negatives and improve accuracy of diagnosis. It utilizes learning from earlier diagnoses and failures to improve the computation time required to find a correct diagnosis. Furthermore, causality-based hybrid planning/prediction is utilized for finding minimum-cardinality diagnoses.

In an accompanying work [12], we integrate our approach to diagnostic reasoning into a plan execution and monitoring framework, and perform further analysis to show the usefulness of diagnostic reasoning and repairs for replanning. We use the ASP formulation of the robotic domain, an ASP-based modification of the planning domain description for diagnosis, and the ASP solvers for hybrid planning and diagnosis. In that sense, the presented work complements the results in [12] by describing the formulations and transformations in action description languages, by performing reasoning tasks in action query languages with the possibility of using SAT solvers, by extending the experimental evaluations to SAT solvers, and by providing some theoretical guarantees over the proposed transformation of the robotic domain description.

**Acknowledgements** Thanks to anonymous reviewers for useful comments. This work is partially supported by TUBITAK Grants 111E116, 113M422 and 114E491 (ChistEra COACHES).

## References

1. Balduccini, M., Gelfond, M.: Diagnostic reasoning with A-Prolog. Theory and Practice of Logic Programming 3(4–5), 425–461 (2003)
2. Baral, C., McIlraith, S., Son, T.C.: Formulating diagnostic problem solving using an action language with narratives and sensing. In: Proc. of KR (2000)
3. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM 54(12), 92–103 (2011)

4. Caldiran, O., Haspalamutgil, K., Ok, A., Palaz, C., Erdem, E., Patoglu, V.: Bridging the gap between high-level reasoning and low-level control. In: Proc. of LPNMR (2009)
5. Casolary, M., Lee, J.: Representing the language of the causal calculator in answer set programming. In: Proc. of ICLP (Technical Communications) (2011)
6. Diankov, R.: Automated Construction of Robotic Manipulation Programs. Ph.D. thesis, Carnegie Mellon University, Robotics Institute (August 2010)
7. Eén, N., Sörensson, N.: An extensible sat-solver. In: Proc. of SAT (2003)
8. Eiter, T., Erdem, E., Faber, W., Senko, J.: A logic-based approach to finding explanations for discrepancies in optimistic plan execution. Fundamenta Informaticae 79, 25–69 (2007)
9. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer-Set Programming. In: Proc. IJCAI (2005)
10. Erdem, E., Haspalamutgil, K., Palaz, C., Patoglu, V., Uras, T.: Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In: Proc. of ICRA (2011)
11. Erdem, E., Haspalamutgil, K., Patoglu, V., Uras, T.: Causality-based planning and diagnostic reasoning for cognitive factories. In: Proc. of ETFA (2012)
12. Erdem, E., Patoglu, V., Saribatur, Z.G.: Integrating hybrid diagnostic reasoning in plan execution monitoring for cognitive factories with multiple robots. In: Proc. of ICRA (2015)
13. Erdem, E., Patoglu, V., Schüller, P.: A systematic analysis of levels of integration between high-level task planning and low-level feasibility checks. In: Proc. of RCRA (2014)
14. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: clasp: A conflict-driven answer set solver. In: Proc. of LPNMR (2007)
15. Gelfond, M., Kahl, Y.: Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach. Cambridge Univ. Press, USA (2014)
16. Gelfond, M., Lifschitz, V.: Action languages. ETAI 2, 193–210 (1998)
17. Giacomo, G.D., Reiter, R., Soutchanski, M.: Execution monitoring of high-level robot programs. In: Proc. of KR (1998)
18. Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., Turner, H.: Nonmonotonic causal theories. Artif. Intell. 153, 49–104 (2004)
19. Iwan, G.: Explaining what went wrong in dynamic domains. In: Proc. of CogRob (2000)
20. de Kleer, J., Williams, B.C.: Diagnosis with behavioral modes. In: Proc. of IJCAI (1989)
21. Kleer, J.D., Mackworth, A.K., Reiter, R.: Characterizing diagnoses and systems. Artif. Intell. 56(2), 197–222 (1992)
22. Lifschitz, V.: Answer set programming and plan generation. Artif. Intell. 138, 39–54 (2002)
23. Lifschitz, V.: What is answer set programming? In: Proc. of AAAI (2008)
24. Marek, V., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In: The Logic Programming Paradigm: a 25-Year Perspective. Springer Verlag (1999)
25. McCain, N.C.: Causality in Commonsense Reasoning about Actions. Ph.D. thesis (1997)
26. McIlraith, S.A.: Explanatory diagnosis: Conjecturing actions to explain observations. In: Proc. of KR (1998)
27. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Annals of Mathematics and Artificial Intelligence 25, 241–273 (1999)
28. Reiter, R.: A theory of diagnosis from first principles (1987)
29. Sohrabi, S., Baier, J.A., McIlraith, S.A.: Diagnosis as planning revisited. In: Proc. KR (2010)
30. Thielscher, M.: A theory of dynamic diagnosis. ETAI 2(11) (1997)
31. Weyhrauch, R.W.: Prolegomena to a theory of formal reasoning. Tech. rep., Stanford University (1978)
32. Zaeh, M., Beetz, M., Shea, K., Reinhart, G., Bender, K., Lau, C., Ostgathe, M., Vogl, W., Wiesbeck, M., Engelhard, M., Ertelt, C., Rhr, T., Friedrich, M., Herle, S.: The cognitive factory. In: Changeable and Reconf. Manufacturing Systems, pp. 355–371 (2009)
33. Zhang, S., Sridharan, M., Gelfond, M., Wyatt, J.L.: Towards an architecture for knowledge representation and reasoning in robotics. In: Proc. of ICSR (2014)