Abstracting Noisy Robot Programs*

Till Hofmann
RWTH Aachen University
Aachen, Germany
hofmann@kbsg.rwth-aachen.de

Vaishak Belle
University of Edinburgh
Edinburgh, United Kingdom
vbelle@ed.ac.uk

ABSTRACT

Abstraction is a commonly used process to represent some low-level system by a more coarse specification with the goal to omit unnecessary details while preserving important aspects. While recent work on abstraction in the situation calculus has focused on non-probabilistic domains, we describe an approach to abstraction of probabilistic and dynamic systems. Based on a variant of the situation calculus with probabilistic belief, we define a notion of bisimulation that allows to abstract a detailed probabilistic basic action theory with noisy actuators and sensors by a possibly non-stochastic basic action theory. By doing so, we obtain abstract Golog programs that omit unnecessary details and which can be translated to detailed programs for execution. This simplifies the implementation of noisy robot programs, opens up the possibility of using non-stochastic reasoning methods (e.g., planning) on probabilistic problems, and provides domain descriptions that are more easily interpretable.

KEYWORDS

Logic; Robot Programs; Noise; Abstraction

ACM Reference Format:
Till Hofmann and Vaishak Belle. 2023. Abstracting Noisy Robot Programs. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 9 pages.

1 INTRODUCTION

Abstraction — the "process of mapping a representation of a problem onto a new representation" [18] — is a ubiquitous concept both in human behavior and in computing systems, e.g., a simple activity such as buying milk involves dozens of actions that a human conveniently abstracts into a single task, and machine instructions (which itself are abstractions of physical processes) are abstracted by higher programming languages. It has also seen widespread usage in several areas of artificial intelligence research [30], in particular in task planning. Abstraction typically involves suppressing irrelevant information and therefore allows reasoning about complex problems that would otherwise be infeasible. In the context of intelligent agents, abstraction typically serves three purposes [6]: (1) it provides a way to structure knowledge, (2) it allows reasoning about larger problems by abstracting the problem domain, resulting in a smaller search space, (3) it may provide more meaningful explanations and is therefore critical for *explainable AI*. The need for abstraction becomes particularly apparent when dealing with robots: as a robot acts in a dynamic environment with imperfect sensors and actuators, its actions are inherently noisy, e.g., it may intend to *move* but may get stuck with some probability. However, when programming such a robot, it is desirable to ignore those probabilistic aspects and instead work with a high-level and non-stochastic system, where the *move* action always succeeds, for all of the reasons above: Correctly designing a probabilistic domain is challenging, reasoning on such a domain is hard, and understanding how such a system operates is difficult. This becomes even more important when considering a robot that may get a hardware upgrade: while the low-level behavior changes (e.g., a new sensor has a different noise profile), the high-level behavior should not be affected. By using abstraction, we only need to update the low-level model and may keep the high-level program as is.

In this paper, we present an abstraction framework for robot programs with probabilistic belief. Starting with the logic $\mathcal{DS}$ [7], a modal variant of the situation calculus with probabilistic belief, we describe a transition semantics for noisy Golog programs in Section 3. Based on this logic, we propose a notion of abstraction of noisy programs, building on top of abstraction of probabilistic static models [6] and non-stochastic dynamic models [2]. We do so by defining a notion of bisimulation of probabilistic dynamic systems in Section 4 and we show that the notions of sound and complete abstraction carry over. We also demonstrate how this abstraction framework can be used to define a high-level domain, where noisy actions are abstracted away and thus, no probabilistic reasoning is necessary. We conclude in Section 5.

2 RELATED WORK

Reasoning about actions. The situation calculus [26, 28] is a logical formalism for reasoning about dynamical domains based on first-order logic. In the situation calculus, world states are represened explicitly as first-order terms called *situations*, where *fluents* describe (possibly changing) properties of the world and actions are axiomatized in basic action theories (BATs). Golog [16, 23] is a programming language based on the situation calculus that allows to control the high-level behavior of robots. $\mathcal{ES}$ [21] is a modal variant and epistemic extension of the situation calculus, where situations are part of the semantics but do not appear as terms in the language. $\mathcal{ESG}$ extends $\mathcal{ES}$ with a transition semantics for Golog programs, which has been used for program verification [11]. The situation calculus, $\mathcal{ES}$, and $\mathcal{ESG}$ are all deterministic and non-stochastic, i.e., the execution of an action always results in a unique successor state. De Giacomo and Lespérance [15] extend the situation calculus with non-deterministic actions, where the environment chooses one of several possible outcomes of an action. Bacchus et al. [1] extend the classical situation calculus with degrees of belief and noisy actions. In a similar fashion, $\mathcal{DS}$ [7] extends $\mathcal{ES}$ with degrees of belief and probabilistic actions, where the environment again may choose

*This is an extended version of the paper to be presented at AAMAS'23.

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom.

an outcome (possibly from an unbounded domain) with some predefined probability, allowing probabilistic representations of robot actions, in particular noisy sensors and actuators. More recently, reasoning about actions in $\mathcal{DS}$ has been shown to be amenable to regression [25] and progression [24] analogous to regression and progression in $\mathcal{ES}$ and the classical situation calculus [28].

Abstraction. Giunchiglia and Walsh [18] define abstraction generally as a mapping between a ground and an abstract formal system, such that the abstract representation preserves desirable properties while omitting unnecessary details to make it simpler to handle. Abstraction has been widely used in several fields of AI [30]. Hierarchical task network (HTN) planning systems such as SHOP2 [27] decompose tasks into subtasks to accomplish some overall objective, which has also been used in the situation calculus [17]. Macro planners such as MacroFF [9] combine action sequences into macro operators to improve planner performance, e.g., by collecting action traces from plan executions on robots [19], or by learning them from training problems [10]. Similarly, Saribatur and Eiter [31] use abstraction in Answer Set Programming to reduce the search space, improving solver performance. Cui et al. [13] leverage abstraction for generalized planning, i.e., for finding general solutions for a set of similar planning problems. Abstraction has also been used to analyze causal models [4, 29]. Of particular interest for this work is the notion of *constructive abstraction* [5], where the refinement mapping partitions the low-level variables such that each cell has a unique corresponding high-level variable. Holtzen et al. [20] describe an abstraction framework for probabilistic programs and also describe an algorithm to generate abstractions. REBA [32] is a framework for robot planning that uses abstract and deterministic ASP programs to determine a course of action, which are then translated to POMDPs for execution. Banihashemi et al. [2] describe a general abstraction framework based on the situation calculus, where a refinement mapping maps a high-level BAT to a low-level BAT and which is capable of online execution with sensing actions [3]. The framework has been used to effectively synthesize plan process controllers in a smart factory scenario [14]. In contrast to this work, they assume non-probabilistic and deterministic actions. On the other hand, Belle [6] defines abstraction in a probabilistic but static propositional language and describes a search algorithm to derive such abstractions. In this paper, we build on the two approaches to obtain abstraction in a probabilistic and dynamic first-order language with an unbounded domain.

3 THE LOGIC $\mathcal{DSG}$

We start by introducing the logic $\mathcal{DSG}$, which we will then use to define abstraction over noisy programs in Section 4. $\mathcal{DSG}$ extends $\mathcal{DS}$ [7] with a transition semantics for Golog, analogous to how $\mathcal{ESG}$ [12] extends $\mathcal{ES}$ [21]. In the same way as $\mathcal{DS}$, the logic uses a countably infinite set of *rigid designators* $\mathcal{R}$, which allows to define quantification substitutionally. Similar to $\mathcal{DS}$, $\mathcal{ES}$, and $\mathcal{ESG}$, it uses a possible-worlds semantics, where a world defines the state of the world not only initially but after any sequence of actions. It uses the modal operator $[\cdot]$ to refer to the state after executing some program, e.g., $[\delta]\alpha$ states that α is true after every possible execution of the program δ. Additionally, it uses the modal operator $\mathbf{B}$ to describe the agent's *belief*, e.g., $\mathbf{B}(Loc(2) : 0.5)$ states the the agent believes with degree 0.5 to be in location 2.

3.1 Syntax

Definition 1 (Symbols of $\mathcal{DSG}$). *The symbols of the language are from the following vocabulary:*

(1) infinitely many variables $x, y, \ldots, u, v, \ldots, a, a_1, \ldots$;
(2) rigid function symbols of every arity, e.g., near, $goto(x, y)$;
(3) fluent predicates of every arity, such as $At(l)$; we assume that this list contains the following distinguished predicates:
- *Poss to denote the executability of an action;*
- *oi to denote that two actions are indistinguishable from the agent's viewpoint; and*
- *l that takes an action as its first argument and the action's likelihood as its second argument;*

(4) connectives and other symbols: $=, \wedge, \neg, \forall, \square, [\cdot], \mathbf{B}$.

Definition 2 (Terms of $\mathcal{DSG}$). *The set of terms of $\mathcal{DSG}$ is the least set such that (1) every variable is a term, (2) if $t_1, \ldots, t_k$ are terms and f is a k-ary function symbol, then $f(t_1, \ldots, t_k)$ is a term.*

As in $\mathcal{DS}$, $\mathcal{R}$ denotes the set of all ground rigid terms. We assume that they contain the rational numbers, i.e., $\mathbb{Q} \subseteq \mathcal{R}$.

Definition 3 (Formulas). *The* formulas of $\mathcal{DSG}$ *are the least set such that*

(1) if $t_1, \ldots, t_k$ are terms and P is a k-ary predicate symbol, then $P(t_1, \ldots, t_k)$ is a formula,
(2) if t_1 and t_2 are terms, then $(t_1 = t_2)$ is a formula,
(3) if α and β are formulas, x is a variable, δ is a program (defined below),[1] and $r \in \mathbb{Q}$, then $\alpha \wedge \beta$, $\neg\alpha$, $\forall x.\, \alpha$, $\square\alpha$, $[\delta]\alpha$, and $\mathbf{B}(\alpha : r)$ are formulas.

We read $\square\alpha$ as "α holds after executing any sequence of actions", $[\delta]\alpha$ as "α holds after the execution of program δ" and $\mathbf{B}(\alpha : r)$ as "α is believed with probability r".[2] We also write $\mathbf{K}\alpha$ for $\mathbf{B}(\alpha : 1)$, to be read as "α is known".[3] We use True as abbreviation for $\forall x(x = x)$ to denote truth. For a formula α, we write α^x_r for the formula resulting from α by substituting every occurrence of x with r. For a finite set of formulas $\Sigma = \{\alpha_1, \ldots, \alpha_n\}$, we may just write Σ for the conjunction $\alpha_1 \wedge \ldots \wedge \alpha_n$, e.g., $\mathbf{K}\Sigma$ for $\mathbf{K}(\alpha_1 \wedge \cdots \wedge \alpha_n)$. A predicate symbol with terms from $\mathcal{R}$ as arguments is called an *atomic formula*, and we denote the set of atomic formulas with $\mathcal{P}$. Furthermore, a formula is called *bounded* if it contains no $\square$ operator, *static* if it contains no $[\cdot]$ or $\square$ operators, *objective* if it contains no $\mathbf{B}$ or $\mathbf{K}$, and *fluent* if it is static and does not mention *Poss*, $\mathbf{B}$, or $\mathbf{K}$.

We define the syntax of programs used by the operator $[\delta]$:

Definition 4 (Programs).

$$\delta ::= t \mid \alpha? \mid \delta_1;\delta_2 \mid \delta_1|\delta_2 \mid \pi x.\, \delta \mid \delta^*$$

where t is a ground rigid term and α is a static formula. A program consists of actions t, tests $\alpha?$, sequences $\delta_1;\delta_2$, nondeterministic

[1] Note that although the definitions of formulas (Definition 3) and programs (Definition 4) mutually depend on each other, they are still well-defined: programs only allow static situation formulas and static situation formulas may not refer to programs.
[2] The original version of the logic also has an only-knowing modal operator $\mathbf{O}$, which captures the idea that something and only that thing is known. For the sake of simplicity, we ignore this operator in our presentation.
[3] We use "knowledge" and "belief" interchangeably, but do not require that knowledge be true in the real world (i.e., weak S5).

branching $\delta_1|\delta_2$, nondeterministic choice of argument $\pi x.\,\delta$, and nondeterministic iteration δ^*.

Note that we do not allow interleaved concurrency $\delta_1\|\delta_2$ known from ConGolog [16].[4] We also use *nil* as abbreviation for True?, the empty program that always succeeds. Similarly to formulas, δ^x_r denotes the program resulting from δ by substituting every x with r. Furthermore, we define:

$$\begin{aligned}\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ fi} &:= (\phi?;\delta_1) \mid (\neg\phi?;\delta_2)\\ \textbf{if } \phi_1 \textbf{ then } \delta_1 \textbf{ elif } \phi_2 \textbf{ then } \delta_2 \textbf{ fi} &:= (\phi_1?;\delta_1) \mid (\neg\phi_1 \wedge \phi_2?;\delta_2)\\ \textbf{while } \phi \textbf{ do } \delta \textbf{ done} &:= (\phi?;\delta)^*;\neg\phi?\end{aligned}$$

3.2 Semantics

As described above, the operator **B** describes the degree of belief. In order to capture noisy actions and sensors, we need to talk about the *likelihood of possible outcomes* as well as the fact that when a noisy action is executed, the intended outcome may not be the same as the desired outcome. The latter is captured using the notion of *observational indistinguishability*. Both likelihood of possible outcomes and observational indistinguishability are built into the worlds using distinguished symbols and then modelled using *basic action theories*, as described in Section 3.3.

Similar to $\mathcal{DS}$, the semantics of $\mathcal{DSG}$ is given in terms of *possible worlds*, where a world defines the truth of each fluent both initially and after any sequence of actions:

Definition 5 (Trace). *A trace $z = \langle a_1, \ldots, a_n\rangle$ is a finite sequence of $\mathcal{R}$. We denote the set of traces as $\mathcal{Z}$ and the empty trace with $\langle\rangle$.*

A world defines the truth of each ground atom from $\mathcal{P}$ not only initially but after any sequence of actions:

Definition 6 (World). *A world is mapping $w : \mathcal{P}\times\mathcal{Z} \to \{0,1\}$. The set of all worlds is denoted as $\mathcal{W}$.*

We require that every world $w \in \mathcal{W}$ defines a unary predicate *Poss*, a binary predicate l that behaves like a function (i.e., there is exactly one $q \in \mathbb{Q}$ such that $w[l(a,q),z] = 1$ for any a, z), as well as an equivalence relation $oi \subseteq \mathcal{R}\times\mathcal{R}$, which define the possibility, the likelihood, and the observational indistinguishability of actions.

We call a pair $(w,z) \in \mathcal{W}\times\mathcal{Z}$ a *state*, we denote the set of all states with $\mathcal{S}$, and we use $S, S_i, \ldots \subseteq \mathcal{S}$ to denote sets of states. Given a state (w,z), the predicate $l(a,q)$ states that the action likelihood of action a in state (w,z) is q. We can inductively apply l to compute the likelihood of a sequence:

Definition 7 (Action Sequence Likelihood). *The action sequence likelihood $l^* : \mathcal{W}\times\mathcal{Z} \to \mathbb{Q}^{\geq 0}$ is defined inductively:*

- *$l^*(w,\langle\rangle) = 1$ for every $w \in \mathcal{W}$,*
- *$l^*(w, z\cdot r) = l^*(w,z)\times q$ where $w[l(r,q),z] = 1$.*

Next, to deal with partially observable states, we define:

Definition 8 (Observational indistinguishability).

(1) Given a world $w \in \mathcal{W}$, we define $\sim_w \subset \mathcal{Z}\times\mathcal{Z}$ inductively:

- *$\langle\rangle \sim_w z'$ iff $z' = \langle\rangle$*
- *$z\cdot r \sim_w z'$ iff $z' = z^*\cdot r^*$, $z \sim_w z^*$, $w[oi(r,r^*),z] = 1$*

(2) We say w is observationally indistinguishable from w', written $w \approx_{oi} w'$ iff for all $a, a' \in \mathcal{R}$, $z \in \mathcal{Z}$: $w[oi(a,a'),z] = w'[oi(a,a'),z]$.

(3) For $w, w' \in \mathcal{W}$, $z, z' \in \mathcal{Z}$, we say (w,z) is observationally indistinguishable from (w',z'), written $(w,z) \approx_{oi} (w',z')$, iff $w \approx_{oi} w'$ and $z \sim_w z'$.

Intuitively, $z \sim_w z'$ means that the agent cannot distinguish whether it executed z or z'. For states, $(w,z) \approx_{\mathrm{oi}} (w',z')$ is to be understood as "if the agent believes to be in state (w,z), it may also be in state (w',z')", i.e., it cannot distinguish the worlds w, w' and traces z, z'. As $\approx_{\mathrm{oi}}$ is an equivalence relation, the set of its equivalence classes on a set of states S induces a partition, which we denote with $S/{\approx_{\mathrm{oi}}}$.

We extend the executability of an action to traces:

Definition 9 (Executable trace). *For a trace z, we define $exec(z)$ inductively:*

- *for $z = \langle\rangle$, $exec(z) :=$ True*
- *for $z = a\cdot z'$, $exec(z) := Poss(a) \wedge [a]exec(z')$*

As in BHL and $\mathcal{DS}$, it is possible to permit the agent to entertain any set of initial distributions. As an example, the initial theory could say that $\mathbf{B}(p:0.5) \vee \mathbf{B}(p:0.6)$, which says that the agent is not sure about the distribution of p. In this case, there would be at least two distributions in the epistemic state e. As another example, if we say $\mathbf{B}(p \vee q : 1)$, then this says that the disjunction is believed with probability 1, but it does not specify the probability of p or q, resulting in infinitely many distributions that are compatible with this constraint. Thus, not committing to a single distribution results in higher expressivity in the representation of uncertainty.

Definition 10 (Compatible States). *Given an epistemic state e, a world w, a trace z, and a formula α, we define the states $S^{e,w,z}_\alpha$ compatible to (e,w,z) wrt to α:*

$$S^{e,w,z}_\alpha = \{(w',z') \mid (w',z') \approx_{oi} (w,z), e, w' \models exec(z') \wedge [z']\alpha\}$$

We write S_α for $S^{e,w,z}_\alpha$ if e, w, z are clear from the context.

To define the semantics of belief, we first define *epistemic states*, which assign probabilities to worlds:

Definition 11 (Epistemic state). *A distribution is a mapping $\mathcal{W} \to \mathbb{R}^{\geq 0}$. An epistemic state is any set of distributions.*

This notion of distribution is not directly a probability distribution. To obtain probability distributions, we define:

Definition 12 (Normalization). *For any distribution d and any set $\mathcal{V} = \{(w_1,z_1),(w_2,z_2),\ldots\}$, we define:*

(1) $\textsc{Bnd}(d,\mathcal{V},r)$ iff there is no k s.t. $\sum_{i=1}^k d(w_i)\times l^*(w_i,z_i) > r$.

(2) $\textsc{Eq}(d,\mathcal{V},r)$ iff $\textsc{Bnd}(d,\mathcal{V},r)$ and there is no $r' < r$ such that $\textsc{Bnd}(d,\mathcal{V},r')$ holds.

(3) For any $\mathcal{U} \subseteq \mathcal{V}$: $\textsc{Norm}(d,\mathcal{U},\mathcal{V},r)$ iff $\exists b \neq 0$ such that $\textsc{Eq}(d,\mathcal{U},b\times r)$ and $\textsc{Eq}(d,\mathcal{V},b)$.

Intuitively, given $\textsc{Norm}(d,\mathcal{U},\mathcal{V},r)$, r can be seen as the normalization of the weights of worlds in $\mathcal{U}$ in relation to the set of worlds $\mathcal{V}$ as accorded by d. The conditions Bnd and Eq are auxiliary conditions to define Norm, where $\textsc{Bnd}(d,\mathcal{V},r)$ states that the weight

[4] The reason will become apparent later on. Intuitively, if we allow interleaved concurrency, then the low-level program could pause the execution of a high-level action and continue with a different high-level action, possibly leading to different effects. This significantly complicates the formal treatment relating the probabilities of high-level worlds to their low-level counterparts.

of worlds in $\mathcal{V}$ is bounded by b and $\text{Eq}(d, \mathcal{V}, r)$ expresses that the weight of worlds in $\mathcal{V}$ is equal to b. Belle et al. [8] have shown that although the set of worlds $\mathcal{W}$ is in general uncountable, this leads to a well-defined summation over the weights of worlds.

To simplify notation, we also write $\text{Norm}(d, \mathcal{U}, \mathcal{V}) = r$ for $\text{Norm}(d, \mathcal{U}, \mathcal{V}, r)$ and $\text{Norm}(d_1, \mathcal{U}_1, \mathcal{V}_1) = \text{Norm}(d_2, \mathcal{U}_2, \mathcal{V}_2)$ if there is an r such that $\text{Norm}(d_1, \mathcal{U}_1, \mathcal{V}_1, r)$ and $\text{Norm}(d_2, \mathcal{U}_2, \mathcal{V}_2, r)$. Finally, we write $\text{Norm}(d, \mathcal{U}_1, \mathcal{V}) + \text{Norm}(d, \mathcal{U}_2, \mathcal{V}) = r$ if there are r_1, r_2 with $\text{Norm}(d, \mathcal{U}_1, \mathcal{V}, r_1)$, $\text{Norm}(d, \mathcal{U}_2, \mathcal{V}, r_2)$, and $r = r_1 + r_2$.

We continue with the program transition semantics, which defines the traces resulting from executing some program. The transition semantics is defined in terms of *configurations* $\langle z, \delta \rangle$, where z is a trace describing the actions executed so far and δ is the remaining program. In some places, the transition semantics refers to the truth of formulas (see Definition 15 below).[5]

Definition 13 (Program Transition Semantics). *The transition relation $\xrightarrow{e,w}$ among configurations, given an epistemic state e and a world w, is the least set satisfying*

(1) $\langle z, a \rangle \xrightarrow{e,w} \langle z \cdot a, nil \rangle$ *if* $w, z \models Poss(a)$
(2) $\langle z, \delta_1; \delta_2 \rangle \xrightarrow{e,w} \langle z \cdot a, \gamma; \delta_2 \rangle$, *if* $\langle z, \delta_1 \rangle \xrightarrow{e,w} \langle z \cdot a, \gamma \rangle$,
(3) $\langle z, \delta_1; \delta_2 \rangle \xrightarrow{e,w} \langle z \cdot a, \delta' \rangle$ *if* $\langle z, \delta_1 \rangle \in \mathcal{F}^{e,w}$ *and* $\langle z, \delta_2 \rangle \xrightarrow{e,w} \langle z \cdot a, \delta' \rangle$
(4) $\langle z, \delta_1 | \delta_2 \rangle \xrightarrow{e,w} \langle z \cdot a, \delta' \rangle$ *if* $\langle z, \delta_1 \rangle \xrightarrow{e,w} \langle z \cdot a, \delta' \rangle$ *or* $\langle z, \delta_2 \rangle \xrightarrow{e,w} \langle z \cdot a, \delta' \rangle$
(5) $\langle z, \pi x.\, \delta \rangle \xrightarrow{e,w} \langle z \cdot a, \delta' \rangle$, *if* $\langle z, \delta^x_r \rangle \xrightarrow{e,w} \langle z \cdot a, \delta' \rangle$ *for some* $r \in \mathcal{R}$
(6) $\langle z, \delta^* \rangle \xrightarrow{e,w} \langle z \cdot a, \gamma; \delta^* \rangle$ *if* $\langle z, \delta \rangle \xrightarrow{e,w} \langle z \cdot a, \gamma \rangle$

The set of final configurations $\mathcal{F}^{e,w}$ is the smallest set s.t.

(1) $\langle z, \alpha? \rangle \in \mathcal{F}^{e,w}$ *if* $e, w, z \models \alpha$,
(2) $\langle z, \delta_1; \delta_2 \rangle \in \mathcal{F}^{e,w}$ *if* $\langle z, \delta_1 \rangle \in \mathcal{F}^{e,w}$ *and* $\langle z, \delta_2 \rangle \in \mathcal{F}^{e,w}$
(3) $\langle z, \delta_1 | \delta_2 \rangle \in \mathcal{F}^{e,w}$ *if* $\langle z, \delta_1 \rangle \in \mathcal{F}^{e,w}$, *or* $\langle z, \delta_2 \rangle \in \mathcal{F}^{e,w}$
(4) $\langle z, \pi x.\, \delta \rangle \in \mathcal{F}^{e,w}$ *if* $\langle z, \delta^x_r \rangle \in \mathcal{F}^{e,w}$ *for some* $r \in \mathcal{R}$
(5) $\langle z, \delta^* \rangle \in \mathcal{F}^{e,w}$

We also write $\xrightarrow{e,w^*}$ for the transitive closure of $\xrightarrow{e,w}$. For a primitive action a, the interpreter may take a transition if a is currently possible. For a sequence of sub-programs $\delta = \delta_1; \delta_2$, the interpreter may take a transition following δ_1, or it may take a transition following δ_2 if δ_1 is final in the current configuration. In the case of nondeterministic branching $\delta_1 | \delta_2$, it may follow the transitions of the first or the second sub-program. For the nondeterministic pick operator $\pi x.\, \delta$, it may follow any transition that results from the program δ^x_r, where x is substituted by some ground term r. Finally, for nondeterministic iteration δ^*, the interpreter may take the same transitions as δ (i.e., continue with another iteration). For the final configurations, atomic tests $\alpha?$ are final if α is satisfied in the current configuration. The sequence of sub-programs $\delta_1; \delta_2$ is final if both sub-programs are final. For nondeterministic branching, the program $\delta_1 | \delta_2$ is final if either sub-program is final. Similarly, for $\pi x.\, \delta$, the program is final if it is final for any substitution of x. Nondeterministic iteration δ^* is final, i.e., the interpreter may always decide to stop (and not continue with the next iteration).

[5] As above, although they depend on each other, the semantics is well-defined, as the transition semantics only refers to static formulas which may not contain programs.

Following the transition semantics for a given program δ, we obtain a set of *program traces*:

Definition 14 (Program Traces). *Given an epistemic state e, a world w, and a trace z, the set $\|\delta\|^z_{e,w}$ of traces of program δ is defined as the following set:*

$$\|\delta\|^z_{e,w} = \{z' \in \mathcal{Z} \mid \langle z, \delta \rangle \xrightarrow{e,w^*} \langle z \cdot z', \delta' \rangle \text{ and } \langle z \cdot z', \delta' \rangle \in \mathcal{F}^{e,w}\}$$

Compared to $\mathcal{ESG}$, this transition semantics also refers to the epistemic state e, as test formulas can also mention belief operators. Additionally, in contrast to $\mathcal{ESG}$, it only allows a transition for an atomic action if the action is possible in the current state. Also, while $\mathcal{ESG}$ allows infinite traces, we only allow finite traces, as we focus on terminating programs.

Finally, we can define the semantics for $\mathcal{DSG}$ formulas:

Definition 15 (Truth of Formulas). *Given an epistemic state e, a world w, and a formula α, we define for every $z \in \mathcal{Z}$:*

(1) $e, w, z \models F(t_1, \ldots, t_k)$ *iff* $w[F(t_1, \ldots, t_k), z] = 1$
(2) $e, w, z \models \mathbf{B}(\alpha : r)$ *iff* $\forall d \in e:\ \text{Norm}(d, S_\alpha, S_{\mathit{TRUE}}, r)$
(3) $e, w, z \models (t_1 = t_2)$ *iff* t_1 *and* t_2 *are identical*
(4) $e, w, z \models \alpha \wedge \beta$ *iff* $e, w, z \models \alpha$ *and* $e, w, z \models \beta$
(5) $e, w, z \models \neg\alpha$ *iff* $e, w, z \not\models \alpha$
(6) $e, w, z \models \forall x.\, \alpha$ *iff* $e, w, z \models \alpha^x_r$ *for all* $r \in \mathcal{R}$.
(7) $e, w, z \models \Box\alpha$ *iff* $e, w, z \cdot z' \models \alpha$ *for all* $z' \in \mathcal{Z}$
(8) $e, w, z \models [\delta]\alpha$ *iff* $e, w, z \cdot z' \models \alpha$ *for all* $z' \in \|\delta\|^z_{e,w}$.

Note in particular that Item 2 states that the degree of belief in a formula is obtained by looking at the normalized weight of the possible worlds that satisfy the formula.

We write $e, w \models \alpha$ for $e, w, \langle\rangle \models \alpha$. Also, if α is objective, we write $w, z \models \alpha$ for $e, w, z \models \alpha$ and $w \models \alpha$ for $w, \langle\rangle \models \alpha$. Additionally, for a set of sentences Σ, we write $e, w, z \models \Sigma$ if $e, w, z \models \phi$ for all $\phi \in \Sigma$, and $\Sigma \models \alpha$ if $e, w \models \Sigma$ entails $e, w \models \alpha$ for every model (e, w).

3.3 Basic Action Theories

A basic action theory (BAT) defines the effects of all actions of the domain, as well as the initial state:

Definition 16 (Basic Action Theory). *Given a finite set of predicates $\mathcal{F}$ including oi and l, a set Σ of sentences only mentioning fluent predicates in $\mathcal{F}$ is called a basic action theory (BAT) over $\mathcal{F}$ iff $\Sigma = \Sigma_0 \cup \Sigma_{pre} \cup \Sigma_{post}$ and*

(1) Σ_0 *is any set of fluent sentences,*
(2) Σ_{pre} *consists of a single sentence of the form* $\Box Poss(a) \equiv \pi$, *where* π *is a fluent formula with free variable* a,[6]
(3) Σ_{post} *is a set of successor state axioms (SSAs) of the form* $\Box Poss(a) \supset ([a]F(\vec{x}) \equiv \gamma_F)$, *one for each fluent predicate* $F \in \mathcal{F}$ *and where* γ_F *is a fluent formula with free variables among* a *and* $\vec{x}$.

Given a BAT Σ, we say that a program δ is a program over Σ if it only mentions fluents and actions from Σ.

[6] We assume that free variables are universally quantified from the outside, $\Box$ has lower syntactic precedence than the logical connectives, and $[\cdot]$ has the highest priority, so that $\Box Poss(a) \equiv \pi$ stands for $\forall a.\, \Box(Poss(a) \equiv \gamma)$ and $\Box Poss(a) \supset ([a]F(\vec{x}) \equiv \gamma_F)$ stands for $\forall a, \vec{x}.\, \Box(Poss(a) \supset ([a]F(\vec{x}) \equiv \gamma_F))$.

Note that the SSAs slightly differ from $\mathcal{ES}$ and $\mathcal{ESG}$, where they have the form $\Box[a]F(\vec{x}) \equiv \gamma_F$. In contrast to $\mathcal{ES}$ and $\mathcal{ESG}$, the SSAs in $\mathcal{DSG}$ only define the effects of an action if the action is currently possible and otherwise do not say anything about the action's effects. This is necessary because we include $Poss(a)$ in the transition semantics (Definition 13). To understand why, consider the following example: if $w, z \models \neg Poss(a)$, then by Definition 15.8, $w, z \models [a]\neg F()$ is vacuously true for any fluent F because there is no trace $z' \in \|a\|^z_{e,w}$, contradicting to a SSA $\Box[a]F() \equiv \gamma_F$. Restricting the SSA to possible actions avoids this issue.[7]

3.3.1 A Noisy BAT. We present a BAT for a simple robotics scenario with noisy actions, inspired from [1, 7]. In this scenario, a robot moves towards a wall and it is equipped with a sonar sensor that can measure the distance to the wall. A BAT Σ_{move} defining this scenario may look as follows:

- A *move* action is possible if the robot moves one step to the back or to the front. A *sonar* action is always possible:

$$\begin{aligned}\Box Poss(a) \equiv\ & \exists x, y(a = move(x, y) \land (x = 1 \lor x = -1)) \\ & \lor \exists z(a = sonar(z))\end{aligned}$$

- After doing action a, the robot is at position x if a is a *move* action that moves the robot to location x, if a is a *sonar* action that measures distance x, or if a is neither of the two actions and the robot was at location x before

$$\begin{aligned}\Box Poss(a) \supset \big([a]Loc(x) \equiv \\ & \exists y, z, (a = move(y, z) \land Loc(l) \land x = l + z) \\ & \lor a = sonar(x) \\ & \lor \neg\exists y, z(a = move(y, z) \lor a = sonar(y)) \land Loc(x)\big)\end{aligned}$$

- For the *sonar* action, the likelihood that the robot measures the correct distance is 0.8, the likelihood that it measures a distance with an error of ± 1 is 0.1. Furthermore, for the *move* action, the likelihood that the robot moves the intended distance x is 0.6, the likelihood that the actual movement y is off by ± 1 is 0.2:

$$\begin{aligned}\Box l(a, u) \equiv \\ & \exists z(a = sonar(z) \land Loc(x) \land u = \Theta(x, z, .8, .1)) \\ & \lor \exists x, y(a = move(x, y) \land u = \Theta(x, y, .6, .2)) \\ & \lor \neg\exists x, y, z(a = move(x, y) \lor a = sonar(z)) \land u = .0\end{aligned}$$

where $\Theta(u, v, c, d) = \begin{cases} c & \text{if } u = v \\ d & \text{if } |u - v| = 1 \\ 0 & \text{otherwise} \end{cases}$.

- The robot cannot detect the distance that it has actually moved, i.e., any actions $move(x, y)$ and $move(x, z)$ are o.i.:

$$\Box oi(a, a') \equiv a = a' \lor \exists x, y, z(a = move(x, y) \land a' = move(x, z))$$

- Initially, it is 3 units away from the wall: $Loc(x) \equiv x = 3$

[7] Claßen [11] proposes a different solution by allowing an action transition even if the action is impossible and then augmenting the program by guarding each action with a test $Poss(a)$?. We prefer the presented solution because the transition semantics only allows actions that are actually possible without augmenting the program.

Based on this BAT, we define a program that first moves the robot close to the wall and then back:[8]

$sonar()$;
while $\neg\mathbf{K}\exists x(Loc(x) \land x \leq 2)$ **do** $move(-1); sonar()$ **done** ;
while $\neg\mathbf{K}\exists x(Loc(x) \land x > 5)$ **do** $move(1); sonar()$ **done**

The robot first measures its distance to the wall and then moves closer until it knows that its distance to the wall is less than two units. Afterwards, it moves away until it knows that is more than five units away from the wall. As the robot's *move* action is noisy, each *move* is followed by *sonar* to measure how far it is away from the wall. One possible execution trace of this program may look as follows:

$$\begin{aligned} z_l = \langle & sonar(3), move(-1, 0), sonar(3), move(-1, -1), \\ & sonar(2), move(-1, -1), sonar(1), move(1, 1), \\ & sonar(3), move(1, 1), sonar(2), move(1, 1), \\ & sonar(4), move(1, 1), sonar(6)\rangle \end{aligned} \tag{1}$$

First, the robot (correctly) senses that it is three units away from the wall and starts moving. However, the first *move* does not have the desired effect: the robot intended to move by one unit but actually did not move (indicated by the second argument being 0). After the second *move*, the robot is at $Loc(2)$, as it started at $Loc(3)$ and moved successfully once. However, as its sensor is noisy and it measured $sonar(2)$, it believes that it could also be at $Loc(3)$. Hence, it executes another *move* and then senses $sonar(1)$, after which it knows for sure that it is at most two units away from the wall. In the second part, the robot moves back until it knows that it is further than five units away from the wall. As this simple example shows, the trace z_l is already quite hard to understand. While it is clear from the BAT what each action does, the robot's intent is not immediately obvious and the trace is cluttered with noise.

3.3.2 An Abstract BAT. We present a second, more abstract BAT for the same scenario without noisy actions:

- Initially, the robot is in the middle: $At(l) \equiv l = mid$
- The robot may *goto* the locations *near* and *far*:[9]

$$\Box Poss(a) \equiv a = goto(near) \lor a = goto(far)$$

- After doing action a, the robot is at location l if a is $goto(l)$ or if a is no *goto* action and the robot has been at l before:

$$\begin{aligned}\Box Poss(a) \supset \\ & \big([a]At(l) \equiv a = goto(l) \lor \neg\exists x(a = goto(x)) \land At(l)\big)\end{aligned}$$

- The action likelihood axiom states that no action is noisy:

$$\begin{aligned}\Box l(a, u) \equiv\ & (a = goto(near) \lor a = goto(far)) \land u = 1.0 \\ & \lor \neg\exists x(a = goto(x)) \land u = 0.0\end{aligned}$$

- The agent can distinguish all actions: $\Box oi(a, a') \equiv a = a'$

In the following, we will connect the low-level BAT Σ_{move} with the high-level BAT Σ_{goto} by using *abstraction.*

[8] The unary $move(x)$ can be understood as abbreviation $move(x) := \pi y\, move(x, y)$, where nature nondeterministically picks the distance y that the robot really moved (similarly for $sonar()$).

[9] For the sake of brevity, we do not allow the robot to go to *mid*.

4 ABSTRACTION

In this section, we define the abstraction of a low-level BAT Σ_l with a high-level BAT Σ_h. This will allow us to construct abstract GOLOG programs over the high-level BAT, which are equivalent and can be translated to some program over the low-level BAT. For the sake of simplicity[10], we assume in the following that an epistemic state e is always a singleton, i.e., $e_h = \{d_h\}$ and $e_l = \{d_l\}$. To translate the high-level BAT Σ_h into the low-level BAT Σ_l, we map Σ_h to Σ_l by mapping each high-level fluent to a low-level formula, and every high-level action to a low-level program:

DEFINITION 17 (REFINEMENT MAPPING). *Given two basic action theories Σ_l over $\mathcal{F}_l$ and Σ_h over $\mathcal{F}_h$. The function m is a* refinement mapping *from Σ_h to Σ_l iff:*

(1) For every action $a(\vec{x})$ mentioned in Σ_h, $m(a(\vec{x})) = \delta_a(\vec{x})$, where $\delta_a(\vec{x})$ is a Golog program over the low-level theory Σ_l with free variables among $\vec{x}$.
(2) For every fluent predicate $F \in \mathcal{F}_h$, $m(F(\vec{x})) = \phi_F(\vec{x})$, where $\phi_F(\vec{x})$ is a static formula over $\mathcal{F}_l$ with free variables among $\vec{x}$.

For a formula α over $\mathcal{F}_h$, we also write $m(\alpha)$ for the formula obtained by applying m to each fluent predicate and action mentioned in α. For a trace $z = \langle a_1, a_2, \ldots \rangle$ of actions from Σ_h, we also write $m(z)$ for $\langle m(a_1), m(a_2), \ldots \rangle$. For a program δ over Σ_h, the program $m(\delta)$ is the same program as δ with each primitive action a replaced by $m(a)$ and each formula α replaced by $m(\alpha)$.

Continuing our example, we define a refinement that maps Σ_{goto} to Σ_{move} by mapping each high-level fluent to a low-level formula and each high-level action to a low-level program:

- The high-level fluent $At(l)$ is mapped to a low-level formula by translating the distance to *near*, *mid*, and *far*:

$$\begin{aligned} At(l) \mapsto\ & l = near \land \exists x(Loc(x) \land x \le 2) \\ & \lor l = mid \land \exists x(Loc(x) \land x > 2 \land x \le 5) \\ & \lor l = far \land \exists x(Loc(x) \land x > 5) \end{aligned}$$

- The action *goto* is mapped to a program that guarantees that the robot reaches the right position:

$goto(x) \mapsto sonar();$
if $x = near$ **then**
while $\neg\mathbf{K}\exists x(Loc(x) \land x \le 2)$ **do** $move(-1); sonar()$ **done**
elif $x = far$ **then**
while $\neg\mathbf{K}\exists x(Loc(x) \land x > 5)$ **do** $move(1); sonar()$ **done fi**

To show that a high-level BAT indeed abstracts a low-level BAT, we first define a notion of isomorphism, intuitively stating that two states satisfy the same fluents:

DEFINITION 18 (OBJECTIVE ISOMORPHISM). *We say (w_h, z_h) is objectively m-isomorphic to (w_l, z_l), written $(w_h, z_h) \sim_m (w_l, z_l)$ iff for every atomic formula α mentioned in Σ_h:*

$$w_h, z_h \models \alpha \textit{ iff } w_l, z_l \models m(\alpha)$$

[10]The technical results do not hinge on this, but allowing arbitrary epistemic states would make the main results and proofs more tedious. For the general case, we need to set up for every distribution on the high level a corresponding distribution on the low level and establish a bisimulation for each of those pairs.

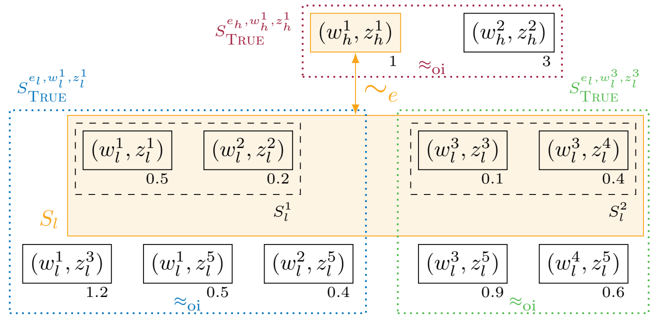

Figure 1: An example for epistemic isomorphism.

Additionally, because we need to relate degrees of belief, we need to connect the two BATs in terms of epistemic states. To do so, we define epistemic isomorphism as follows:

DEFINITION 19 (EPISTEMIC ISOMORPHISM). *For every $(w_h, z_h) \in \mathcal{S}$ and $S_l \subseteq \mathcal{S}$, we say that (d_h, w_h, z_h) is epistemically m-isomorphic to (d_l, S_l), written $(d_l, w_h, z_h) \sim_e (d_l, S_l)$ iff for the partition $P = S_l/{\approx_{oi}}$, for each $S_l^i \in P$ and $(w_l^i, z_l^i) \in S_l^i$:*

$$\textit{Norm}(d_h, \{(w_h, z_h)\}, S_{\textit{TRUE}}^{e_h, w_h, z_h}) = \textit{Norm}(d_l, S_l^i, S_{\textit{TRUE}}^{e_l, w_l^i, z_l^i})$$

The intuition of epistemic isomorphism is illustrated in Figure 1: As the high-level state (w_h^1, z_h^1) is more abstract than the low-level states S_l, multiple low-level states (highlighted in orange) may be isomorphic to the same high-level state. Hence, each high-level state is mapped to a set of low-level states. To be epistemically isomorphic, they must entail the same beliefs, so the corresponding normalized weights must be equal. However, we do not require the low-level states S_l to be observationally indistinguishable. Indeed, since we have a high-level action corresponding to many low-level actions, low-level states are typically not observationally indistinguishable. Thus, we partition S_l according to $\approx_{\text{oi}}$, obtaining S_l^1 and S_l^2 and we require the normalized weight of (w_h, z_h) (in relation to the compatible states $S_{\text{TRUE}}^{e_h, w_h^1, z_h^1}$) to be the same as the normalized weight of each member S_l^i of the partition (in relation to the corresponding compatible states, $S_{\text{TRUE}}^{e_l, w_l^1, z_l^1}$ and $S_{\text{TRUE}}^{e_l, w_l^3, z_l^3}$ respectively).

Having established objective and epistemic isomorphisms, we can now define a suitable notion of bisimulation:

DEFINITION 20 (BISIMULATION).
*A relation $B \subseteq \mathcal{S} \times \mathcal{S}$ is an m-*bisimulation between (e_h, w_h) and (e_l, w_l) *if $((w_h, z_h), (w_l, z_l)) \in B$ implies that*

(1) $(w_h, z_h) \sim_m (w_l, z_l)$,
(2) $(d_h, w_h, z_h) \sim_e \left(d_l, \{(w_l', z_l') \mid ((w_h, z_h), (w_l', z_l')) \in B\}\right)$,
(3) $w_h \models exec(z_h)$ and $w_l \models exec(z_l)$,
(4) for every high-level action a, if $w_h, z_h \models Poss(a)$, then there is $z_l' \in \|m(a)\|_{e_l, w_l}^{z_l}$ such that $((w_h, z_h \cdot a), (w_l, z_l \cdot z_l')) \in B$,
(5) for every high-level action a, if there is $z_l' \in \|m(a)\|_{e_l, w_l}^{z_l}$, then $w_h, z_h \models Poss(a)$ and $((w_h, z_h \cdot a), (w_l, z_l \cdot z_l')) \in B$,

(6) for every (w'_h, z'_h) with $(w'_h, z'_h) \approx_{oi} (w_h, z_h)$, $d_h(w'_h) > 0$, and $e_h, w'_h \models exec(z'_h)$, there is $(w'_l, z'_l) \approx_{oi} (w_l, z_l)$ such that $((w'_h, z'_h), (w'_l, z'_l)) \in B$,
(7) for every (w'_l, z'_l) with $(w'_l, z'_l) \approx_{oi} (w_l, z_l)$, $d_l(w'_l) > 0$, and $e_l, w'_l \models exec(z'_l)$, there is $(w'_h, z'_h) \approx_{oi} (w_h, z_h)$ such that $((w'_h, z'_h), (w'_l, z'_l)) \in B$.

We call a bisimulation B definite *if $((w_h, z_h), (w_l, z_l)) \in B$ and $((w'_h, z'_h), (w_l, z_l)) \in B$ implies $(w_h, z_h) = (w'_h, z'_h)$.*

We say that (e_h, w_h) is bisimilar to (e_l, w_l) relative to refinement mapping m, written $(e_h, w_h) \sim_m (e_l, w_l)$, if and only if there exists a definite m-bisimulation relation B between (e_h, w_h) and (e_l, w_l) such that $((w_h, \langle\rangle), (w_l, \langle\rangle)) \in B$.

The general idea of bisimulation is that two states are bisimilar if they have the same local properties (i.e., they are isomorphic) and each reachable state from the first state has a corresponding reachable state from the second state (and vice versa) such that the two successors are again bisimilar. Here, properties 1, 2, and 3 refer to static properties of (w_h, z_h) and (w_l, z_l). While property 1 establishes objective isomorphism of (w_h, z_h) and (w_l, z_l), property 2 establishes epistemic isomorphism between (w_h, z_h) and all states (w'_l, z'_l) that occur in B. As usual in bisimulations, we also require that if we follow a high-level transition of the system, there is a corresponding low-level transition (and vice versa). Here, such a transition may be an action that is executed (properties 4 and 5), or it may be an epistemic transition from the current state to another observationally indistinguishable state (properties 6 and 7).

Our notion of bisimulation is similar to bisimulation for abstracting non-stochastic and objective basic action theories [2]. In comparison, the notion of objective isomorphism (property 1) and reachable states via actions (properties 4 and 5) are analogous, while epistemic isomorphism (property 2) and reachable states via observational indistinguishability (properties 6 and 7) have no corresponding counterparts.

Given a corresponding m-bisimulation, we want to show that (e_h, w_h) is a model of a formula α iff (e_l, w_l) is a model of the mapped formula $m(\alpha)$. To do so, we first show that this is true for static formulas, not considering programs. In the second step, we will show that the high-level and low-level models induce the same program traces, which will then allow us to extend the statement to bounded formulas, which may refer to programs. We start with static formulas:[11]

Theorem 1. *Let $(e_h, w_h) \sim_m (e_l, w_l)$ with definite m-bisimulation B. For every static formula α and every pair of traces z_h, z_l with $((w_h, z_h), (w_l, z_l)) \in B$:*

$$e_h, w_h, z_h \models \alpha \text{ iff } e_l, w_l, z_l \models m(\alpha)$$

Proof Idea. By structural induction on α. The interesting case is $\alpha = \mathbf{B}(\beta : r)$. Let

$$\textsc{Norm}(d_h, S_\beta^{e_h, w_h, z_h}, S_{\text{True}}^{e_h, w_h, z_h}) = r_h$$
$$\textsc{Norm}(d_l, S_{m(\beta)}^{e_l, w_l, z_l}, S_{\text{True}}^{e_l, w_l, z_l}) = r_l$$

We need to show that $r_h = r_l$.
$\leq$: Let $(w_h^i, z_h^i) \in S_\beta^{e_h, w_h, z_h}$. We can ignore those (w_h^i, z_h^i) with $d_h(w_h^i) = 0$ because they do not contribute to r_h. By Definition 20.6, there is a (w_l^i, z_l^i) with $((w_h^i, z_h^i), (w_l^i, z_l^i)) \in B$ and $(w_l^i, z_l^i) \approx_{\text{oi}} (w_l, z_l)$. From Definition 20.2 and Definition 19, we know that for each such (w_h^i, z_h^i), (w_h^i, z_h^i) is epistemically isomorphic to the union S_l of all bisimilar (w'_l, z'_l), i.e., $(d_h, w_h^i, z_h^i) \sim_e (d_l, S_l)$, where $S_l = \{(w'_l, z'_l) \mid ((w_h^i, z_h^i), (w'_l, z'_l)) \in B\}$. Using the partition $P = S_l/{\approx_{\text{oi}}}$, there is $S_l^i \in P$ with $(w_l^i, z_l^i) \in S_l^i$. It follows:

$$\textsc{Norm}(d_h, \{(w_h^i, z_h^i)\}, S_{\text{True}}^{e_h, w_h^i, z_h^i}) = \textsc{Norm}(d_l, S_l^i, S_{\text{True}}^{e_l, w_l, z_l})$$

As B is definite, we can directly take the union of both sides to obtain the overall probability of $S_\beta^{e_h, w_h, z_h}$:

$$\textsc{Norm}(d_h, S_\beta^{e_h, w_h, z_h}, S_{\text{True}}^{e_h, w_h, z_h}) = \textsc{Norm}(d_l, \bigcup_i S_l^i, S_{\text{True}}^{e_l, w_l, z_l})$$

Furthermore, by induction, for each $(w'_l, z'_l) \in S_l^i$, it follows that $e_l, w'_l \models [z'_l]m(\beta)$ and therefore, $S_l^i \subseteq S_{m(\beta)}^{e_l, w_l, z_l}$. With that,

$$\textsc{Norm}(d_l, \bigcup_i S_l^i, S_{\text{True}}^{e_l, w_l, z_l}) \leq \textsc{Norm}(d_l, S_{m(\beta)}^{e_l, w_l, z_l}, S_{\text{True}}^{e_l, w_l, z_l})$$

Thus, $r_h \leq r_l$.
$\geq$: For each $(w_l^i, z_l^i) \in S_{m(\beta)}^{e_l, w_l, z_l}$ with $d_l(w_l^i) > 0$, by Definition 20.7, there is a (w_h^i, z_h^i) with $((w_h^i, z_h^i), (w_l^i, z_l^i)) \in B$. As above, with Definition 20.2, this (w_h^i, z_h^i) is epistemically isomorphic to the union S_l of all bisimilar (w'_l, z'_l). Let $P = S_l/{\approx_{\text{oi}}}$ and $S_l^i \in P$ with $(w_l^i, z_l^i) \in S_l^i$. It can be shown that

$$\textsc{Norm}(d_l, \bigcup_i S_l^i, S_{\text{True}}^{e_l, w_l, z_l}) = \textsc{Norm}(d_h, \bigcup_i \left\{(w_h^i, z_h^i)\right\}, S_{\text{True}}^{e_h, w_h, z_h})$$

We can partition $S_{m(\beta)}^{e_l, w_l, z_l}$ into $\{S_{m(\beta)}^1, S_{m(\beta)}^1, \ldots\}$ such that for each i, $S_{m(\beta)}^i \subseteq S_l^i$. Clearly,

$$\textsc{Norm}(d_l, \bigcup_i S_{m(\beta)}^i, S_{\text{True}}^{e_l, w_l, z_l}) \leq \textsc{Norm}(d_l, \bigcup_i S_l^i, S_{\text{True}}^{e_l, w_l, z_l})$$

Finally, by induction, $e_h, w_h^i \models [z_h^i]\beta$, thus $(w_h^i, z_h^i) \in S_\beta^{e_h, w_h, z_h}$, and therefore $\bigcup_i \{(w_h^i, z_h^i)\} \subseteq S_\beta^{e_h, w_h, z_h}$. We obtain:

$$\textsc{Norm}(d_l, S_{m(\beta)}^{e_l, w_l, z_l}, S_{\text{True}}^{e_l, w_l, z_l}) \leq \textsc{Norm}(d_h, S_\beta^{e_h, w_h, z_h}, S_{\text{True}}^{e_h, w_h, z_h})$$

Thus, $r_h \geq r_l$. □

Using Theorem 1, we show that if (e_h, w_h) is bisimilar to (e_l, w_l), then they induce the same traces of a program δ:

Lemma 1. *Let $(e_h, w_h) \sim_m (e_l, w_l)$ with m-bisimulation B such that $((w_h, z_h), (w_l, z_l)) \in B$ and let δ be an arbitrary program.*
(1) If $z'_l \in \|m(\delta)\|_{e_l, w_l}^{z_l}$ is a low-level trace, then there is a high-level trace $z'_h \in \|\delta\|_{e_h, w_h}^{z_h}$ such that $z'_h = \langle a_1, \ldots, a_n\rangle$, $z'_l = \langle m(a_1), \ldots, m(a_n)\rangle$, and $(z_h \cdot z'_h, z_l \cdot z'_l) \in B$.
(2) If $z'_h = \langle a_1, \ldots, a_n\rangle \in \|\delta\|_{e_h, w_h}^{z_h}$ is a high-level trace, then there is a low-level trace $z'_l \in \|m(\delta)\|_{e_l, w_l}^{z_l}$ such that $z'_l = \langle m(a_1), \ldots, m(a_n)\rangle$ and $(z_h \cdot z'_h, z_l \cdot z'_l) \in B$.

Note that Lemma 1 would not hold if δ contained interleaved concurrency. Intuitively, this is because for a high-level program such as $a_h^1 \| a_h^2$, the only valid high-level traces would be $\langle a_h^1, a_h^2\rangle$ and $\langle a_h^2, a_h^1\rangle$, i.e., one action is completely executed before the

[11] All proofs can be found in the appendix.

other action is started. On the other hand, with $m(a_h^1) = a_l^1; a_l^2$ and $m(a_h^2) = a_l^3; a_l^4$, we may obtain interleaved traces such as $\langle a_l^1, a_l^3, a_l^2, a_l^4 \rangle$, which does not have a corresponding high-level trace.[12]

With Lemma 1, we extend Theorem 1 to bounded formulas:

Theorem 2. *Let $(e_h, w_h) \sim_m (e_l, w_l)$. For all bounded formulas α and traces z_h, z_l with $(z_h, z_l) \in B$:*

$$e_h, w_h, z_h \models \alpha \text{ iff } e_l, w_l, z_l \models m(\alpha)$$

It directly follows that the high- and low-level models entail the same formulas after executing some program δ:

Corollary 1. *Let $(e_h, w_h) \sim_m (e_l, w_l)$. Then for any high-level Golog program δ and static high-level formula β:*

$$e_l, w_l \models [m(\delta)]m(\beta) \text{ iff } e_h, w_h \models [\delta]\beta$$

4.1 Sound and Complete Abstraction

In the previous section, we described properties of abstraction with respect to particular models (e_h, w_h) and (e_l, w_l). However, we are usually more interested in the relationship between a high-level BAT Σ_h and a low-level BAT Σ_l:[13]

Definition 21 (Sound Abstraction). *We say that Σ_h is a* sound abstraction *of Σ_l relative to refinement mapping m if and only if for each model $(e_l, w_l) \models \mathbf{K}\Sigma_l \land \Sigma_l$, there exists a model $(e_h, w_h) \models \mathbf{K}\Sigma_h \land \Sigma_h$ such that $(e_h, w_h) \sim_m (e_l, w_l)$.*

Conclusions by Σ_h are consistent with Σ_l:

Theorem 3. *Let Σ_h be a sound abstraction of Σ_l relative to mapping m. Then, for every bounded formula α, if $\mathbf{K}\Sigma_h \land \Sigma_h \models \alpha$, then $\mathbf{K}\Sigma_l \land \Sigma_l \models m(\alpha)$.*

While a sound abstraction ensures that any entailment of the high-level BAT Σ_h is consistent with the low-level BAT Σ_l, the Σ_h may have less information than Σ_l, e.g., Σ_h may consider it possible that some program δ is executable, while Σ_l knows that it is not. This leads to a second notion of abstraction:

Definition 22 (Complete Abstraction). *We say that Σ_h is a* complete abstraction *of Σ_l relative to refinement mapping m if and only if for each model $(e_h, w_h) \models \mathbf{K}\Sigma_h \land \Sigma_h$, there exists a model $(e_l, w_l) \models \mathbf{K}\Sigma_l \land \Sigma_l$ such that $(e_h, w_h) \sim_m (e_l, w_l)$.*

Indeed, if we have a complete abstraction, then Σ_h must entail everything that Σ_l entails:

Theorem 4. *Let Σ_h be a complete abstraction of Σ_l relative to mapping m. Then, for every bounded formula α, if $\mathbf{K}\Sigma_l \land \Sigma_l \models m(\alpha)$, then $\mathbf{K}\Sigma_h \land \Sigma_h \models \alpha$.*

The strongest notion is the combination of both:

Definition 23 (Sound and Complete Abstraction). *We say that Σ_h is a* sound and complete abstraction *of Σ_l relative to refinement mapping m if Σ_h is both a sound and a complete abstraction of Σ_l wrt m.*

[12] While a limited form of concurrency could be permitted by only allowing interleaved execution of high-level actions (i.e., each $m(a)$ must be completely executed before switching to a different branch of execution), we omit this for the sake of simplicity.

[13] Notice that we require the real world to have the same physical laws as that believed by the agent, which is fairly standard. We do not require the agent knows everything about the real world, nor that the agent beliefs are also true in the real world.

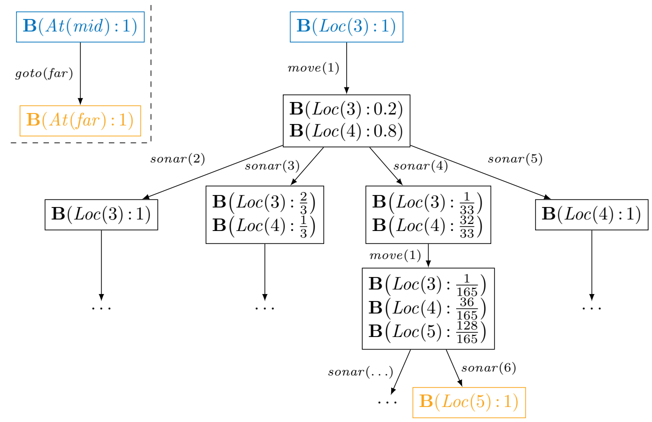

Figure 2: Bisimulation for the running example, where sets of states are summarized by the belief that they entail.

Theorem 5. *Let Σ_h be a sound and complete abstraction of Σ_l relative to refinement mapping m. Then, for every bounded formula α, $\mathbf{K}\Sigma_h \land \Sigma_h \models \alpha$ iff $\mathbf{K}\Sigma_l \land \Sigma_l \models m(\alpha)$.*

Coming back to our example, we can show that Σ_{goto} is indeed a sound and complete abstraction of Σ_{move}:

Proposition 1. *Σ_{goto} is a sound and complete abstraction of Σ_{move} relative to refinement mapping m.*

Figure 2 shows an exemplary bisimulation for the running example. The single transition for *goto* of the high-level BAT is shown on the left. The agent knows that it is initially in the middle and after doing *goto*(*far*), it is far away from the wall. Some corresponding transitions of the low-level BAT are shown on the right: Initially, the agent knows that it is at *Loc*(3), which is a bisimilar state to the initial high-level state (blue). Eventually, it reaches a state where it knows that it is at *Loc*(5), which is again a bisimilar state to the corresponding high-level state (orange).

With Theorem 5, it follows that both BATs entail the same (mapped) formulas. Therefore, we can use Σ_{goto} for reasoning and planning, e.g., we may write a high-level Golog program in terms of Σ_{goto} and then use a classical Golog interpreter to find a ground action sequence that realizes the program. To continue the example, we may write a very simple abstract program δ_h that first moves to the wall if necessary and then moves back:

$$\textbf{if } \neg At(near) \textbf{ then } goto(near) \textbf{ fi}\,;\, goto(far)$$

If the robot is initially not near the wall (as in our example), the following sequence is a realization of the program:

$$\langle goto(near), goto(far) \rangle$$

This high-level trace is much simpler than the trace of the low-level program shown in Equation 1. At the same time, as Σ_{goto} is a sound and complete abstraction of Σ_{move}, this sequence may be translated to Σ_{move} by applying the refinement mapping m and the translated program then takes care of noisy sensors and actuators.

5 CONCLUSION

In this paper, we have presented a framework for abstraction of probabilistic dynamic domains. More specifically, in a first step, we have defined a transition semantics for Golog programs with noisy actions based on $\mathcal{DS}$, a variant of the situation calculus with probabilistic belief. We have then defined a suitable notion of bisimulation in the logic that allows the abstraction of noisy robot programs in terms of a refinement mapping from a high-level to a low-level basic action theory. This abstraction method allows to obtain a significantly simpler high-level domain, which can be used for reasoning or high-level programming without the need to deal with stochastic actions. Furthermore, the resulting programs and traces are much easier to understand, because they do not contain noisy actions and are often much shorter.

While abstractions need to be manually constructed, future work may explore abstraction generation algorithms based on [6, 20]. A further extension might be to provide conditions under which we can modify the low-level program, e.g., with new sensors with different error profiles, without modifying the high-level program.

Interestingly, as the logics $\mathcal{DS}$ and $\mathcal{ES}$ are fully compatible for non-probabilistic formulas not mentioning noisy actions [7] and abstraction allows to get rid of probabilistic formulas and noisy actions, we may construct $\mathcal{ES}$ programs that are sound and complete abstractions of $\mathcal{DS}$ programs. This is a step towards cognitive robotics as envisioned by Reiter [22], where the classical non-probabilistic situation calculus machinery may prove entirely sufficient to define the behavior and termination of real-world robots.

ACKNOWLEDGMENTS

Till was partly supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 2236/1 and the EU ICT-48 2020 project TAILOR (No. 952215). Part of this work was created during a research visit of Till at the University of Edinburgh, which was funded by the German Academic Exchange Service (DAAD). Vaishak was partly supported by a Royal Society University Research Fellowship, UK, and partly supported by a grant from the UKRI Strategic Priorities Fund, UK to the UKRI Research Node on Trustworthy Autonomous Systems Governance and Regulation (EP/V026607/1, 2020–2024).

REFERENCES

[1] Fahiem Bacchus, Joseph Y. Halpern, and Hector J. Levesque. 1999. Reasoning about Noisy Sensors and Effectors in the Situation Calculus. *Artificial Intelligence* 111, 1 (July 1999), 171–208. https://doi.org/10.1016/S0004-3702(99)00031-4

[2] Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. 2017. Abstraction in Situation Calculus Action Theories. In *Proceedings of the 31st Conference on Artificial Intelligence (AAAI)*. AAAI Press, 1048–1055.

[3] Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. 2018. Abstraction of Agents Executing Online and Their Abilities in the Situation Calculus. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*. 1699–1706. https://doi.org/10.24963/ijcai.2018/235

[4] Bita Banihashemi, Shakil Mahmud Khan, and Mikhail Soutchanski. 2022. From Actions to Programs as Abstract Actual Causes. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, 5470–5478.

[5] Sander Beckers and Joseph Y. Halpern. 2019. Abstracting Causal Models. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2678–2685. https://doi.org/10.1609/aaai.v33i01.33012678

[6] Vaishak Belle. 2020. Abstracting Probabilistic Models: Relations, Constraints and Beyond. *Knowledge-Based Systems* 199 (July 2020), 105976. https://doi.org/10.1016/j.knosys.2020.105976

[7] Vaishak Belle and Gerhard Lakemeyer. 2017. Reasoning about Probabilities in Unbounded First-Order Dynamical Domains. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI Press, 828–836.

[8] Vaishak Belle, Gerhard Lakemeyer, and Hector J. Levesque. 2016. A First-Order Logic of Probability and Only Knowing in Unbounded Domains. In *Proceedings of the 30th Conference on Artificial Intelligence (AAAI 2016)*. AAAI Press, 893–899.

[9] Adi Botea, Markus Enzenberger, Martin Müller, and Jonathan Schaeffer. 2005. Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators. *Journal of Artificial Intelligence Research* 24 (2005), 581–621. https://doi.org/10.1613/jair.1696

[10] Lukáš Chrpa, Mauro Vallati, and Thomas Leo McCluskey. 2014. MUM: A Technique for Maximising the Utility of Macro-operators by Constrained Generation and Use. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press.

[11] Jens Claßen. 2013. *Planning and Verification in the Agent Language Golog*. Ph.D. Dissertation. RWTH Aachen University.

[12] Jens Claßen and Gerhard Lakemeyer. 2008. A Logic for Non-Terminating Golog Programs. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR)*. AAAI Press, 589–599.

[13] Zhenhe Cui, Yongmei Liu, and Kailun Luo. 2021. A Uniform Abstraction Framework for Generalized Planning. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI Press, 1837–1844.

[14] Giuseppe De Giacomo, Paolo Felli, Brian Logan, Fabio Patrizi, and Sebastian Sardiña. 2022. Situation Calculus for Controller Synthesis in Manufacturing Systems with First-Order State Representation. *Artificial Intelligence* 302 (Jan. 2022), 103598. https://doi.org/10.1016/j.artint.2021.103598

[15] Giuseppe De Giacomo and Yves Lespérance. 2021. The Nondeterministic Situation Calculus. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, Vol. 18. AAAI Press, 216–226. https://doi.org/10.24963/kr.2021/21

[16] Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. 2000. ConGolog, a Concurrent Programming Language Based on the Situation Calculus. *Artificial Intelligence* 121 (2000), 109–169.

[17] Alfredo Gabaldon. 2002. Programming Hierarchical Task Networks in the Situation Calculus. In *AIPS Workshop on Online Planning and Scheduling*. 18.

[18] Fausto Giunchiglia and Toby Walsh. 1992. A Theory of Abstraction. *Artificial Intelligence* 57, 2 (Oct. 1992), 323–389. https://doi.org/10.1016/0004-3702(92)90021-O

[19] Till Hofmann, Tim Niemueller, and Gerhard Lakemeyer. 2017. Initial Results on Generating Macro Actions from a Plan Database for Planning on Autonomous Mobile Robots. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS)*, Vol. 27. AAAI Press, 498–503.

[20] Steven Holtzen, Guy van den Broeck, and Todd Millstein. 2018. Sound Abstraction and Decomposition of Probabilistic Programs. In *Proceedings of the 35th International Conference on Machine Learning (PMLR)*, Vol. 80. ML Research Press, 1999–2008.

[21] Gerhard Lakemeyer and Hector J. Levesque. 2011. A Semantic Characterization of a Useful Fragment of the Situation Calculus with Knowledge. *Artificial Intelligence* 175, 1 (Jan. 2011), 142–164. https://doi.org/10.1016/j.artint.2010.04.005

[22] Hector J. Levesque and Gerhard Lakemeyer. 2008. Cognitive Robotics. In *Foundations of Artificial Intelligence*. Vol. 3. Elsevier, 869–886. https://doi.org/10.1016/S1574-6526(07)03023-4

[23] Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. 1997. GOLOG: A Logic Programming Language for Dynamic Domains. *Journal of Logic Programming* 31, 1-3 (1997), 59–83. https://doi.org/10.1016/S0743-1066(96)00121-5

[24] Daxin Liu and Qihui Feng. 2021. On the Progression of Belief. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, Vol. 18. AAAI Press, 465–474. https://doi.org/10.24963/kr.2021/44

[25] Daxin Liu and Gerhard Lakemeyer. 2021. Reasoning about Beliefs and Meta-Beliefs by Regression in an Expressive Probabilistic Action Logic. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*, Vol. 2. AAAI Press, 1951–1958. https://doi.org/10.24963/ijcai.2021/269

[26] John McCarthy. 1963. *Situations, Actions, and Causal Laws*. Technical Report. Stanford University. 11 pages.

[27] Dana Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. 2003. SHOP2 : An HTN Planning System. *Journal of Artificial Intelligence Research* 20 (2003), 379–404.

[28] Raymond Reiter. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.

[29] Paul K. Rubenstein, Sebastian Weichwald, Stephan Bongers, Joris M. Mooij, Dominik Janzing, Moritz Grosse-Wentrup, and Bernhard Schölkopf. 2017. Causal Consistency of Structural Equation Models. In *Proceedings of the 33rd Annual Conference on Uncertainty in Artificial Intelligence (UAI)*. AUAI.

[30] Lorenza Saitta and Jean-Daniel Zucker. 2013. Abstraction in Artificial Intelligence. In *Abstraction in Artificial Intelligence and Complex Systems*. Springer, 49–63. https://doi.org/10.1007/978-1-4614-7052-6_3

[31] Zeynep G. Saribatur and Thomas Eiter. 2021. Omission-Based Abstraction for Answer Set Programs. *Theory and Practice of Logic Programming* 21, 2 (March 2021), 145–195. https://doi.org/10.1017/S1471068420000095

[32] Mohan Sridharan, Michael Gelfond, Shiqi Zhang, and Jeremy Wyatt. 2019. REBA: A Refinement-Based Architecture for Knowledge Representation and Reasoning in Robotics. *Journal of Artificial Intelligence Research* 65 (June 2019), 87–180. https://doi.org/10.1613/jair.1.11524

PROOFS

THEOREM 1. *Let $(e_h, w_h) \sim_m (e_l, w_l)$ with definite m-bisimulation B. For every static formula α and every pair of traces z_h, z_l with $((w_h, z_h), (w_l, z_l)) \in B$:*

$$e_h, w_h, z_h \models \alpha \text{ iff } e_l, w_l, z_l \models m(\alpha)$$

PROOF OF THEOREM 1. By structural induction on α.

- Let α be an atomic formula. Then, since $(z_h, z_l) \in B$, it follows from Definition 20.1, that $(w_h, z_h) \sim_m (w_l, z_l)$, and thus $w_h, z_h \models \alpha$ iff $w_l, z_l \models m(\alpha)$.
- Let $\alpha = \beta \wedge \gamma$. The claim follows directly by induction and the semantics of conjunction.
- Let $\alpha = \neg\beta$. The claim follows directly by induction and the semantics of negation.
- Let $\alpha = \forall x. \beta$. The claim follows directly by induction and the semantics of all-quantification.
- Let $\alpha = \mathbf{B}(\beta : r)$. By definition, $e_h, w_h \models \mathbf{B}(\beta : r_h)$ iff

$$\textsc{Norm}\Big(d_h, S_\beta^{e_h, w_h, z_h}, S_{\textsc{True}}^{e_h, w_h, z_h}, r_h\Big)$$

Similarly, $e_l, w_l \models \mathbf{B}(m(\beta) : r_l)$ iff

$$\textsc{Norm}\Big(d_l, S_{m(\beta)}^{e_l, w_l, z_l}, S_{\textsc{True}}^{e_l, w_l, z_l}, r_l\Big)$$

$\underline{r_h \leq r_l}$: For each $(w_h^i, z_h^i) \in S_\beta^{e_h, w_h, z_h}$ with $d_h(w_h^i) > 0$ and $e_h, w_h^i \models exec(z_h^i)$, by Definition 20.6, there is a (w_l^i, z_l^i) with $((w_h^i, z_h^i), (w_l^i, z_l^i)) \in B$ and $(w_l^i, z_l^i) \approx_{\text{oi}} (w_l, z_l)$. By Definition 20.2,

$$(d_h, w_h^i, z_h^i) \sim_e \\ (d_l, \underbrace{\{(w_l', z_l') \mid ((w_h^i, z_h^i), (w_l', z_l')) \in B\}}_{=:S_B})$$

Let P be the partition $P = S_B/{\approx_{\text{oi}}}$ of S_B. As $(w_l^i, z_l^i) \in S_B$, there is a $S_l^i \in P$ with $(w_l^i, z_l^i) \in S_l^i$. By Definition 19:

$$\textsc{Norm}(d_h, \{(w_h^i, z_h^i)\}, S_{\textsc{True}}^{e_h, w_h^i, z_h^i}) \\ = \textsc{Norm}(d_l, S_l^i, S_{\textsc{True}}^{e_l, w_l^i, z_l^i})$$

With $(w_l, z_l) \approx_{\text{oi}} (w_l^i, z_l^i)$, it follows that $S_{\textsc{True}}^{e_l, w_l^i, z_l^i} = S_{\textsc{True}}^{e_l, w_l, z_l}$. Hence:

$$\textsc{Norm}(d_h, \{(w_h^i, z_h^i)\}, S_{\textsc{True}}^{e_h, w_h^i, z_h^i}) \\ = \textsc{Norm}(d_l, S_l^i, S_{\textsc{True}}^{e_l, w_l, z_l}) \quad (2)$$

So far, we have only considered $(w_h^i, z_h^i) \in S_\beta^{e_h, w_h, z_h}$ with $d_h(w_h^i) > 0$ and $e_h, w_h^i \models exec(z_h^i)$. By definition of $\textsc{Norm}$, any (w_h', z_h') with $d_h(w_h') = 0$ cannot add to $\textsc{Norm}$. Also, again by definition, for every $(w_h', z_h') \in S_\beta^{e_h, w_h, z_h}$, $e_h, w_h' \models exec(z_h')$. Therefore:

$$\textsc{Norm}(d_h, \bigcup_i \{(w_h^i, z_h^i)\}, S_{\textsc{True}}^{e_h, w_h, z_h}) \\ = \textsc{Norm}(d_h, S_\beta^{e_h, w_h, z_h}, S_{\textsc{True}}^{e_h, w_h, z_h}) \quad (3)$$

Now, as B is definite, it follows for each $i \neq j$ that $S_l^i \neq S_l^j$ and as P is a partition, $S_l^i \cap S_l^j = \emptyset$. With this and with Equation 2 and Equation 3, it follows that

$$\textsc{Norm}(d_h, S_\beta^{e_h, w_h, z_h}, S_{\textsc{True}}^{e_h, w_h, z_h}) \\ = \textsc{Norm}(d_l, \bigcup_i S_l^i, S_{\textsc{True}}^{e_l, w_l, z_l})$$

We continue by showing the connection between all S_l^i and $S_{m(\beta)}^{e_l, w_l, z_l}$: For each $(w_l', z_l') \in S_l^i$, by definition of S_l^i, we have $(w_l', z_l') \approx_{\text{oi}} (w_l, z_l)$. As $((w_h^i, z_h^i), (w_l', z_l')) \in B$, by Definition 20.3, $e_l, w_l' \models exec(z_l')$. Also, it follows by induction that $e_l, w_l', z_l' \models m(\beta)$. Thus, $(w_l', z_l') \in S_{m(\beta)}^{e_l, w_l, z_l}$ and therefore, $S_l^i \subseteq S_{m(\beta)}^{e_l, w_l, z_l}$. Therefore:

$$\textsc{Norm}(d_l, \bigcup_i S_l^i, S_{\textsc{True}}^{e_l, w_l, z_l}) \\ \leq \textsc{Norm}(d_l, S_{m(\beta)}^{e_l, w_l, z_l}, S_{\textsc{True}}^{e_l, w_l, z_l})$$

We summarize:

$$\begin{aligned} r_h &= \textsc{Norm}(d_h, S_\beta^{e_h, w_h, z_h}, S_{\textsc{True}}^{e_h, w_h, z_h}) \\ &= \textsc{Norm}(d_l, \bigcup_i S_l^i, S_{\textsc{True}}^{e_l, w_l, z_l}) \\ &\leq \textsc{Norm}(d_l, S_{m(\beta)}^{e_l, w_l, z_l}, S_{\textsc{True}}^{e_l, w_l, z_l}) \\ &= r_l \end{aligned}$$

Thus, $r_h \leq r_l$.

$\underline{r_l \leq r_h}$: For each $(w_l^i, z_l^i) \in S_{m(\beta)}^{e_l, w_l, z_l}$ with $d_l(w_l^i) > 0$ and $e_l, w_l^i \models exec(z_l^i)$, as $(w_l^i, z_l^i) \approx_{\text{oi}} (w_l, z_l)$, by Definition 20.7, there is a (w_h^i, z_h^i) with $((w_h^i, z_h^i), (w_l^i, z_l^i)) \in B$ and therefore, by Definition 20.2,

$$(d_h, w_h^i, z_h^i) \sim_e \\ (d_l, \underbrace{\{(w_l', z_l') \mid ((w_h^i, z_h^i), (w_l', z_l')) \in B\}}_{=:S_B^i})$$

Let P be the partition $P = S_B^i/{\approx_{\text{oi}}}$ of S_B^i. As $(w_l^i, z_l^i) \in S_B^i$, there is a $S_l^i \in P$ with $(w_l^i, z_l^i) \in S_l^i$. By Definition 19,

$$\textsc{Norm}(d_l, S_l^i, S_{\textsc{True}}^{e_l, w_l^i, z_l^i}) \\ = \textsc{Norm}(d_h, \{(w_h^i, z_h^i)\}, S_{\textsc{True}}^{e_h, w_h^i, z_h^i}) \quad (4)$$

Now, as $(w_h, z_h) \approx_{\text{oi}} (w_h^i, z_h^i)$, it follows that $S_{\textsc{True}}^{e_h, w_h^i, z_h^i} = S_{\textsc{True}}^{e_h, w_h, z_h}$, similarly $S_{\textsc{True}}^{e_l, w_l^i, z_l^i} = S_{\textsc{True}}^{e_l, w_l, z_l}$. Therefore, we can also write Equation 4 as

$$\textsc{Norm}(d_l, S_l^i, S_{\textsc{True}}^{e_l, w_l, z_l}) \\ = \textsc{Norm}(d_h, \{(w_h^i, z_h^i)\}, S_{\textsc{True}}^{e_h, w_h, z_h}) \quad (5)$$

Now, suppose there is j, k with $j \neq k$ such that $(w_h^j, z_h^j) = (w_h^k, z_h^k)$. Clearly, $S_B^j = S_B^k$. Also, $(w_l^j, z_l^j) \in S_{m(\beta)}^{e_l, w_l, z_l}$ and $(w_l^k, z_l^k) \in S_{m(\beta)}^{e_l, w_l, z_l}$, $(w_l^j, z_l^j) \approx_{\text{oi}} (w_l, z_l)$, $(w_l^k, z_l^k) \approx_{\text{oi}} (w_l, z_l)$, and therefore also $(w_l^j, z_l^j) \approx_{\text{oi}} (w_l^k, z_l^k)$. Thus, $S_l^j = S_l^k$. As Equation 5 holds for each i, it follows that

$$\textsc{Norm}(d_l, \bigcup_i S_l^i, S_{\textsc{True}}^{e_l,w_l,z_l}) = \textsc{Norm}(d_h, \bigcup_i \{(w_h^i, z_h^i)\}, S_{\textsc{True}}^{e_h,w_h,z_h}) \quad (6)$$

Let $Q = \{S^1_{m(\beta)}, S^2_{m(\beta)}, \ldots\}$ be the partition of $S^{e_l,w_l,z_l}_{m(\beta)} \cap \{(w_l', z_l') \mid d_l(w_l') > 0\}$ such that $S^i_{m(\beta)} \subseteq S^i_l$. With Equation 5, it directly follows that

$$\begin{aligned}&\textsc{Norm}(d_l, \bigcup_i S^i_{m(\beta)}, S_{\textsc{True}}^{e_l,w_l,z_l})\\ &\leq \textsc{Norm}(d_l, \bigcup_i S_l^i, S_{\textsc{True}}^{e_l,w_l,z_l})\\ &= \textsc{Norm}(d_h, \bigcup_i \{(w_h^i, z_h^i)\}, S_{\textsc{True}}^{e_h,w_h,z_h}) \quad (7)\end{aligned}$$

By definition of Norm, any (w_l', z_l') with $d_l(w_l') = 0$ cannot add to Norm, i.e.,

$$\textsc{Norm}(d_l, \bigcup_i S^i_{m(\beta)}, S_{\textsc{True}}^{e_l,w_l,z_l}) = \textsc{Norm}(d_l, S^{e_l,w_l,z_l}_{m(\beta)}, S_{\textsc{True}}^{e_l,w_l,z_l})$$

With that, Equation 7 can be written as:

$$\textsc{Norm}(d_l, S^{e_l,w_l,z_l}_{m(\beta)}, S_{\textsc{True}}^{e_l,w_l,z_l}) \leq \textsc{Norm}(d_h, \bigcup_i \{(w_h^i, z_h^i)\}, S_{\textsc{True}}^{e_h,w_h,z_h}) \quad (8)$$

Finally, as $((w_h^i, z_h^i), (w_l^i, z_l^i)) \in B$ and $e_l, w_l^i, z_l^i \models m(\beta)$, it follows by induction that $e_h, w_h^i, z_h^i \models \beta$. Therefore, with $(w_h^i, z_h^i) \approx_{\text{oi}} (w_h, z_h)$, we have $(w_h^i, z_h^i) \in S_\beta^{e_h,w_h,z_h}$. Hence:

$$\textsc{Norm}(d_l, S^{e_l,w_l,z_l}_{m(\beta)}, S_{\textsc{True}}^{e_l,w_l,z_l}) \leq \textsc{Norm}(d_h, S_\beta^{e_h,w_h,z_h}, S_{\textsc{True}}^{e_h,w_h,z_h})$$

Therefore $r_l \leq r_h$.

With $r_h = r_l = r$, it follows that $e_h, w_h, z_h \models \mathbf{B}(\beta : r)$ iff $e_l, w_l, z_l \models \mathbf{B}(m(\beta) : r)$.

□

Lemma 1. *Let $(e_h, w_h) \sim_m (e_l, w_l)$ with m-bisimulation B such that $((w_h, z_h), (w_l, z_l)) \in B$ and let δ be an arbitrary program.*

(1) If $z_l' \in \|m(\delta)\|^{z_l}_{e_l,w_l}$ is a low-level trace, then there is a high-level trace $z_h' \in \|\delta\|^{z_h}_{e_h,w_h}$ such that $z_h' = \langle a_1, \ldots, a_n\rangle$, $z_l' = \langle m(a_1), \ldots, m(a_n)\rangle$, and $(z_h \cdot z_h', z_l \cdot z_l') \in B$.

(2) If $z_h' = \langle a_1, \ldots, a_n\rangle \in \|\delta\|^{z_h}_{e_h,w_h}$ is a high-level trace, then there is a low-level trace $z_l' \in \|m(\delta)\|^{z_l}_{e_l,w_l}$ such that $z_l' = \langle m(a_1), \ldots, m(a_n)\rangle$ and $(z_h \cdot z_h', z_l \cdot z_l') \in B$.

Proof of Lemma 1.

(1) By structural induction on δ.

- Let $\delta = a$ and thus $z_l' = \langle m(a)\rangle$. Then, by Definition 20.3, $w_h, z_h \models Poss(a)$, therefore $\langle a\rangle \in \|\delta\|^{z_h}_{e_h,w_h}$ and also $(z_h \cdot a, z_l \cdot z_l') \in B$.
- Let $\delta = \alpha?$. From $z_l' \in \|m(\delta)\|^{z_l}_{e_l,w_l}$, it directly follows that $\langle z_l, m(\alpha)?\rangle \in \mathcal{F}^{e_l,w_l}$, $z_l' = \langle\rangle$ and $e_l, w_l, z_l \models m(\alpha)$. By Theorem 1, it follows that $e_h, w_h, z_h \models \alpha$. Thus, $\langle z_h, \alpha?\rangle \in \mathcal{F}^{e_h,w_h}$, and therefore, for $z_h' = \langle\rangle$, we can follow that $z_h' \in \|\delta\|^{z_h}_{e_h,w_h}$. Finally, as $z_h = z_l = \langle\rangle$ and $(z_h, z_l) \in B$, it follows that $(z_h \cdot z_h', z_l \cdot z_l') \in B$.
- Let $\delta = \delta_1; \delta_2$. By induction, for $z_l^1 \in \|m(\delta_1)\|^{z_l}_{e_l,w_l}$, there is $z_h^1 = \langle a_1, \ldots, a_k\rangle \in \|\delta_1\|^{z_h}_{e_h,w_h}$ with $z_l^1 = \langle m(a_1), \ldots, m(a_k)\rangle$ and $(z_h \cdot z_h^1, z_l, \cdot z_l^1) \in B$. Let $z_l^2 \in \|m(\delta_2)\|^{z_l \cdot z_l^1}_{e_l,w_l}$. It follows again by induction that there is $z_h^2 = \langle a_{k+1}, \ldots, a_n\rangle \in \|\delta_2\|^{z_h \cdot z_h^1}_{e_h,w_h}$ and such that $z_l^2 = \langle m(a_{k+1}), \ldots m(a_n)\rangle$ and $(z_h \cdot z_h^1 \cdot z_h^2, z_l \cdot z_l^1 \cdot z_l^2) \in B$.
- Let $\delta = \delta_1 | \delta_2$. Two cases:
  (a) $z_l' \in \|m(\delta_1)\|^{z_l}_{e_l,w_l}$. Then, by induction, there is $z_h' \in \|\delta_1\|^{z_h}_{e_h,w_h}$ with $z_h' = \langle a_1, \ldots, a_n\rangle$ and such that $z_l' = \langle m(a_1), \ldots, m(a_n)\rangle$ with $(z_h \cdot z_h', z_l \cdot z_l') \in B$.
  (b) $z_l' \in \|m(\delta_2)\|^{z_l}_{e_l,w_l}$. Then, by induction, there is $z_h' \in \|\delta_2\|^{z_h}_{e_h,w_h}$ with $z_h' = \langle a_1, \ldots, a_n\rangle$ and such that $z_l' = \langle m(a_1), \ldots, m(a_n)\rangle$ with $(z_h \cdot z_h', z_l \cdot z_l') \in B$.

(2) By structural induction on δ.

- Let $\delta = a$ and thus $z_h' = \langle a\rangle \in \|\delta\|^{z_h}_{e_h,w_h}$. Therefore, $e_h, w_h, z_h \models Poss(a)$ and thus, by Definition 20, there is $z_l' \in \|m(a)\|^{z_l}_{e_l,w_l}$ with $(z_h \cdot z_h', z_l \cdot z_l') \in B$.
- Let $\delta = \alpha?$. From $z_h' \in \|\delta\|^{z_h}_{e_h,w_h}$, it directly follows that $\langle z_h, a?\rangle \in \mathcal{F}^{e_h,w_h}$, $z_h' = \langle\rangle$, and $e_h, w_h, z_h \models \alpha$. By Theorem 1, it follows that $e_l, w_l, z_l \models m(\alpha)$. Thus, $\langle z_l, m(\alpha)?\rangle \in \mathcal{F}^{e_l,w_l}$, and therefore $z_l' = \langle\rangle \in \|m(\delta)\|^{z_l}_{e_l,w_l}$. Finally, as $z_h = z_l = \langle\rangle$ and $(z_h, z_l) \in B$, it follows that $(z_h \cdot z_h', z_l \cdot z_l') \in B$.
- Let $\delta = \delta_1; \delta_2$. By induction, for $z_h^1 = \langle a_1, \ldots, a_k\rangle \in \|\delta_1\|^{z_h}_{e_h,w_h}$, there is $z_l^1 \in \|m(\delta_1)\|^{z_l}_{e_l,w_l}$ such that $z_l^1 = \langle m(a_1), \ldots, m(a_k)\rangle$ with $(z_h \cdot z_h^1, z_l \cdot z_l^1) \in B$. Again by induction, for $z_h^2 = \langle a_{k+1}, \ldots a_n\rangle \in \|\delta_2\|^{z_h \cdot z_h^1}_{e_h,w_h}$, there is $z_l^2 \in \|m(\delta_2)\|^{z_l \cdot z_l^1}_{e_l,w_l}$ such that $(z_h \cdot z_h^1 \cdot z_h^2, z_l \cdot z_l^1 \cdot z_l^2) \in B$.
- Let $\delta = \delta_1 | \delta_2$. Two cases:
  (a) $z_h' = \langle a_1, \ldots, a_n\rangle \in \|\delta_1\|^{z_h}_{e_h,w_h}$. Then, by induction, there is $z_l' \in \|m(\delta_1)\|^{z_l}_{e_l,w_l}$ with $z_l' = \langle m(a_1), \ldots, m(a_n)\rangle$, and $(z_h \cdot z_h', z_l \cdot z_l') \in B$.
  (b) $z_h' = \langle a_1, \ldots, a_n\rangle \in \|\delta_2\|^{z_h}_{e_h,w_h}$. Then, by induction, there is $z_l' \in \|m(\delta_2)\|^{z_l}_{e_l,w_l}$ with $z_l' = \langle m(a_1), \ldots, m(a_n)\rangle$ and $(z_h \cdot z_h', z_l \cdot z_l') \in B$.

□

Theorem 2. *Let $(e_h, w_h) \sim_m (e_l, w_l)$. For all bounded formulas α and traces z_h, z_l with $(z_h, z_l) \in B$:*

$$e_h, w_h, z_h \models \alpha \textit{ iff } e_l, w_l, z_l \models m(\alpha)$$

Proof of Theorem 2. By structural induction on α.

- Let α be an atomic formula. Then, since $(z_h, z_l) \in B$, we know that $(w_h, z_h) \sim_m (w_l, z_l)$, and thus $w_h, z_h \models \alpha$ iff $w_l, z_l \models m(\alpha)$.
- Let $\alpha = \mathbf{B}(\beta : r)$. Same proof as in Theorem 1.

- Let $\alpha = \beta \wedge \gamma$. The claim follows directly by induction and the semantics of conjunction.
- Let $\alpha = \neg\beta$. The claim follows directly by induction and the semantics of negation.
- Let $\alpha = \forall x.\, \beta$. The claim follows directly by induction and the semantics of all-quantification.
- Let $\alpha = [\delta]\beta$.
  ⇐: Let $e_h, w_h, z_h \not\models [\delta]\beta$. There is a finite trace $z'_h \in \|\delta\|^{z_h}_{e_h,w_h}$ with $e_h, w_h, z_h \cdot z'_h \not\models \beta$. By Lemma 1, there is $z'_l \in \|m(\delta)\|^{z_l}_{e_l,w_l}$ with $(z_h \cdot z'_h, z_l \cdot z'_l) \in B$. By induction, $e_l, w_l, z_l \cdot z'_l \not\models \beta$, and thus $e_l, w_l, z_l \not\models [m(\delta)]m(\beta)$.
  ⇒: Let $(e_l, w_l) \not\models [m(\delta)]m(\beta)$, i.e., there is a finite trace $z'_l \in \|m(\delta)\|^{z_l}_{e_l,w_l}$ with $e_l, w_l, z_l \cdot z'_l \not\models m(\beta)$. By Lemma 1, there is a $z'_h \in \|\delta\|^{z_h}_{e_h,w_h}$ with $(z_h \cdot z'_h, z_l \cdot z'_l) \in B$. By induction, $e_h, w_h, z_h \cdot z'_h \not\models \beta$ and thus $e_h, w_h, z_h \not\models [\delta]\beta$.

□

Corollary 1. *Let $(e_h, w_h) \sim_m (e_l, w_l)$. Then for any high-level Golog program δ and static high-level formula β:*

$$e_l, w_l \models [m(\delta)]m(\beta) \text{ iff } e_h, w_h \models [\delta]\beta$$

Proof of Corollary 1. This is a special case of Theorem 2 with $z_h = \langle\rangle, z_l = \langle\rangle, \alpha = [\delta]\beta$. □

Theorem 3. *Let Σ_h be a sound abstraction of Σ_l relative to mapping m. Then, for every bounded formula α, if $\mathbf{K}\Sigma_h \wedge \Sigma_h \models \alpha$, then $\mathbf{K}\Sigma_l \wedge \Sigma_l \models m(\alpha)$.*

Proof of Theorem 3. Let $\mathbf{K}\Sigma_h \wedge \Sigma_h \models \alpha$. Suppose $\mathbf{K}\Sigma_l \wedge \Sigma_l \not\models m(\alpha)$, i.e., there is a model (e_l, w_l) of $\mathbf{K}\Sigma_l \wedge \Sigma_l$ with $e_l, w_l \not\models m(\alpha)$. As Σ_h is a sound abstraction of Σ_l, there is a model (e_h, w_h) of $\mathbf{K}\Sigma_h \wedge \Sigma_h$ with $(e_h, w_h) \sim_m (e_l, w_l)$. By Theorem 2, $e_h, w_h \not\models \alpha$. Contradiction to $\mathbf{K}\Sigma_h \wedge \Sigma_h \models \alpha$. Thus, $\mathbf{K}\Sigma_l \wedge \Sigma_l \models m(\alpha)$. □

Theorem 4. *Let Σ_h be a complete abstraction of Σ_l relative to mapping m. Then, for every bounded formula α, if $\mathbf{K}\Sigma_l \wedge \Sigma_l \models m(\alpha)$, then $\mathbf{K}\Sigma_h \wedge \Sigma_h \models \alpha$.*

Proof of Theorem 4. Let $\mathbf{K}\Sigma_l \wedge \Sigma_l \models m(\alpha)$. Suppose $\mathbf{K}\Sigma_h \wedge \Sigma_h \not\models \alpha$, i.e., there is a model (e_h, w_h) of $\mathbf{K}\Sigma_h \wedge \Sigma_h$ with $(e_h, w_h) \not\models \alpha$. As Σ_h is a complete abstraction of Σ_l, there is a model (e_l, w_l) with $e_l, w_l \models \mathbf{K}\Sigma_l \wedge \Sigma_l$ and $(e_h, w_h) \sim_m (e_l, w_l)$. By Theorem 2, $e_l, w_l \not\models m(\alpha)$. Contradiction to $\Sigma_l \models m(\alpha)$. Thus, $\mathbf{K}\Sigma_h \wedge \Sigma_h \models \alpha$. □

Theorem 5. *Let Σ_h be a sound and complete abstraction of Σ_l relative to refinement mapping m. Then, for every bounded formula α, $\mathbf{K}\Sigma_h \wedge \Sigma_h \models \alpha$ iff $\mathbf{K}\Sigma_l \wedge \Sigma_l \models m(\alpha)$.*

Proof of Theorem 5. Follows directly from Theorem 3 and Theorem 4 . □

Proposition 1. *Σ_{goto} is a sound and complete abstraction of Σ_{move} relative to refinement mapping m.*

Proof of Proposition 1.

Sound abstraction: We first show that Σ_{goto} is a sound abstraction of Σ_{move}: Let $e_l, w_l \models \mathbf{K}\Sigma_{move} \wedge \Sigma_{move}$. We show by construction that there is a model (e_h, w_h) with $e_h, w_h \models \mathbf{K}\Sigma_{goto} \wedge \Sigma_{goto}$ and $(e_h, w_h) \sim_m (e_l, w_l)$.

First, note that from $e_l, w_l \models \mathbf{K}\Sigma_{move}$, it follows that $d_l(w'_l) > 0$ implies $w'_l \models \Sigma_{move}$. Let $w_h \models \Sigma_{goto}$ and let e_h be an epistemic state such that $d_h(w_h) = 1$ and $d_h(w'_h) = 0$ for every $w'_h \neq w_h$. Clearly, $e_h, w_h \models \mathbf{K}\Sigma_{goto} \wedge \Sigma_{goto}$. Now, let

$$B_0 = \{((w_h, \langle\rangle), (w_l, \langle\rangle))\} \cup \Big\{((w_h, \langle\rangle), (w'_l, \langle\rangle)) \mid d_l(w'_l) > 0\Big\}$$

Next, let

$$B_{i+1} = \Big\{((w'_h, z'_h \cdot a), (w'_l, z'_l \cdot z''_l)) \mid ((w'_h, z'_h), (w'_l, z'_l)) \in B_i, w'_h, z'_h \models Poss(a), z''_l \in \|m(a)\|^{z'_l}_{e_l,w_l}\Big\}$$

As B only mentions a single high-level world w_h, it directly follows that B is definite. We show by induction on i that $B = \bigcup_i B_i$ is an m-bismulation between (e_h, w_h) and (e_l, w_l). Let $((w'_h, z'_h), (w'_l, z'_l)) \in B_i$.

Base case. Note that $z'_h = z'_l = \langle\rangle$ by definition of B_0. We show that all criteria of Definition 20 are satisfied:

1. By definition, $w'_l \models \Sigma_{move}$ and thus $w'_l \models Loc(x) \equiv x = 3$. At the same time, $w'_h \models \Sigma_{goto}$ and thus $w'_h \models At(l) \equiv l = mid$. Therefore, for all l, $w'_h \models At(l)$ iff $w'_l \models m(At(l))$ and thus, $(w'_h, \langle\rangle) \sim_m (w'_l, \langle\rangle)$.
2. By definition of e_h, $\textsc{Norm}(d_h, \{(w_h, \langle\rangle)\}, S^{e_h,w_h,z_h}_{\textsc{True}}, 1)$. Also, $d_l(w'_l) > 0$ iff $((w_h, \langle\rangle), (w'_l, \langle\rangle)) \in B$. Let $S_l = \{(w'_l, \langle\rangle) \mid d_l(w'_l) > 0\}$. It directly follows that for each set S^i_l of the partition $S_l/{\approx_{\text{oi}}}$, $\textsc{Norm}(d_l, S^i_l, S^{e_l,[S^{(i)}_l]}_{\textsc{True}}, 1)$. Thus, $(d_h, w_h, \langle\rangle) \sim_e (d_l, S_l)$.
3. As $z'_h = z'_l = \langle\rangle$, it directly follows that $e_h, w'_h \models exec(z'_h)$ and $e_l, w'_l \models exec(z'_l)$.
4. Let $w'_h \models Poss(a)$. Then, $a = goto(l)$ for some $l \in \{near, far\}$. As $e_l, w'_l \models \Sigma_{move}$, it follows for each such l that there is some $z''_l \in \|m(goto(l))\|^{\langle\rangle}_{e_l,w'_l}$:
   - For $l = near$, $z''_l = \langle sonar(), move(-1,-1), sonar()\rangle$.
   - For $l = far$,
     $z''_l = \langle sonar, move(1,1), sonar(), move(1,1), sonar()\rangle$.

   By definition of B_{i+1}, we obtain $((w'_h, z'_h \cdot a), (w'_l, z'_l \cdot z''_l)) \in B$.
5. Let $z''_l \in \|m(a)\|^{z'_l}_{e_l,w_l}$. Clearly, $a = goto(l)$ for some $l \in \{near, far\}$. By definition of Σ_{goto}, it directly follows that $e_h, w'_h, z'_h \models Poss(a)$. By definition of B_{i+1}, it also follows that $((w'_h, z'_h \cdot a), (w'_l, z'_l \cdot z''_l)) \in B$.
6. Let $(w''_h, z''_h) \approx_{\text{oi}} (w'_h, z'_h)$ with $d_h(w''_h) > 0$ and $e_h, w''_h \models exec(z''_h)$. As $d_h(w''_h) = 0$ for every $w''_h \neq w'_h$ and $z'_h = \langle\rangle$, it directly follows that $(w''_h, z''_h) = (w'_h, z'_h)$ and thus $((w''_h, z''_h), (w'_l, z'_l)) \in B$.
7. Let $(w''_l, z''_l) \approx_{\text{oi}} (w'_l, z'_l)$ with $d_l(w''_l) > 0$ and $e_h, w''_l \models exec(z''_l)$. Clearly, $z''_l = z'_l = \langle\rangle$. By definition of B_0, $((w_h, \langle\rangle), (w''_l, \langle\rangle)) \in B$.

Induction step.

(1) Let $z'_h = z''_h \cdot a$ and $z'_l = z''_l \cdot z'''_l$ for some $z'''_l \in \|m(a)\|^{z''_l}_{e_l,w'_l}$. By construction, $((w'_h, z''_h), (w'_l, z''_l)) \in B$. By induction, $(w'_h, z''_h) \sim_m (w'_l, z''_l)$. Furthermore, $w'_h \models \Sigma_{goto}$ and $w'_l \models \Sigma_{move}$. As before, $a = goto(l)$ with $l \in \{near, far\}$. As $w'_h \models \Sigma_{goto}$, for every l', $e_h, w'_h, z'_h \models At(l')$ iff $l' = l$. Similarly, by definition of m and Σ_{move}, for every l', $e_l, w'_l, z'_l \models m(At(l'))$ iff $l' = l$. Thus, $(w'_h, z'_h) \sim_m (w'_l, z'_l)$.

(2) Suppose

$$(d_h, w'_h, z'_h) \not\sim_e (d_l, \underbrace{\{(w''_l, z''_l) \mid ((w'_h, z'_h), (w''_l, z''_l)) \in B\}}_{=:S_B})$$

First, note that $\textsc{Norm}(d_h, \{(w'_h, z'_h)\}, S^{e_h,w'_h,z'_h}_{\textsc{True}}, 1)$. Therefore, there is a $S^i_l \in S_B/{\approx_{\text{oi}}}$ and $(w^i_l, z^i_l) \in S^i_l$ where $\textsc{Norm}(d_l, S^i_l, S^{e_l,w^i_l,z^i_l}_{\textsc{True}}, 1) \neq 1$, i.e., there is $(w''_l, z''_l) \in S^{e_l,w^i_l,z^i_l}_{\textsc{True}} \setminus S^i_l$ with $d_l(w''_l) \times l^*(w''_l, z''_l) > 0$. It follows that $(w''_l, z''_l) \approx_{\text{oi}} (w^i_l, z^i_l)$ for some $(w^i_l, z^i_l) \in S^i_l$. But then, by definition of Σ_{move}, z''_l is the same as z^i_l, except a possibly different second parameter of each $move(x, y)$ action. Also, $z^i_l = z^{i,1}_l \cdot z^{i,2}_l$, where $z^{i,2}_l \in \|m(a)\|^{z^{i,1}_l}_{e_l,w^i_l}$ for some action a. As $z''_l \sim_{w''_l} z^i_l$, it follows that $z''_l = z''^1_l \cdot z''^2_l$ with $z''^2_l \in \|m(a)\|^{z''^1_l}_{e_l,w''_l}$ and $z''^1_l \sim_{w''_l} z^{i,1}_l$. But then, by induction, there is some (w''_h, z''^1_h) such that $((w''_h, z''^1_h), (w''_l, z''^1_l)) \in B$, and therefore, by definition of B, also $((w''_h, z''^1_h \cdot a), (w''_l, z''_l)) \in B$. Contradiction to $(w''_l, z''_l) \in S^{e_l,w^i_l,z^i_l}_{\textsc{True}} \setminus S^i_l$. It follows:

$$(d_h, w'_h, z'_h) \sim_e (d_l, \underbrace{\{(w''_l, z''_l) \mid ((w'_h, z'_h), (w''_l, z''_l)) \in B\}}_{=:S_B})$$

(3) $w'_h \models exec(z'_h)$ and $w'_l \models exec(z'_l)$ directly follows by construction of B.

(4) Let $w'_h, z'_h \models Poss(a)$. Then, $a = goto(l)$ for some $l \in \{near, far\}$. As $e_l, w'_l \models \Sigma_{move}$, it follows for each such l that there is some $z''_l \in \|m(goto(l))\|^{z'_l}_{e_l,w'_l}$. By definition of B_{i+1}, it also follows that $((w'_h, z'_h \cdot a), (w'_l, z'_l \cdot z''_l)) \in B$.

(5) Let $z''_l \in \|m(a)\|^{z'_l}_{e_l,w_l}$. Clearly, $a = goto(l)$ for some $l \in \{near, far\}$. By definition of Σ_{goto}, it directly follows that $e_h, w'_h, z'_h \models Poss(a)$. By definition of B_{i+1}, it also follows that $((w'_h, z'_h \cdot a), (w'_l, z'_l \cdot z''_l)) \in B$.

(6) Let $(w''_h, z''_h) \approx_{\text{oi}} (w'_h, z'_h)$ with $d_h(w''_h) > 0$ and $e_h, w''_h \models exec(z''_h)$. As $d_h(w''_h) = 0$ for every $w''_h \neq w'_h$, it follows that $w''_h = w'_h$. Furthermore, by definition of Σ_{goto}, $z''_h \sim_{w''_h} z'_h$ iff $z''_h = z'_h$, therefore $(w''_h, z''_h) = (w'_h, z'_h)$ and thus $((w''_h, z''_h), (w'_l, z'_l)) \in B$.

(7) Let $(w''_l, z''_l) \approx_{\text{oi}} (w'_l, z'_l)$ with $d_l(w''_l) > 0$ and $e_h, w''_l \models exec(z''_l)$. As $z''_l \sim_{w''_l} z'_l$, the trace z''_l must consist of the same actions as z'_l, except for a possibly different second parameter in each $move(x, y)$. Furthermore, as Σ_{goto} only contains the action $goto$, the trace z'_l only consists of mapped $goto$ actions, i.e., $z'_l \in \|m(goto(l_1)); \ldots; m(goto(l_n))\|^{\langle\rangle}_{e_l,w'_l}$ We can split $z'_l = z'^1_l \cdot z'^2_l$ such that $z'^2_l \in \|m(goto(l_n))\|^{z'^1_l}_{e_l,w'_l}$. Then, because of $z'' \sim_{w''_l} z'_l$, we can also split z''_l such that $z''_l = z''^1_l \cdot z''^2_l$, $z''^1_l \sim_{w''_l} z'^1_l$ with $z''^2_l \in \|m(goto(l'_n))\|^{z''^1_l}_{e_l,w''_l}$. By induction, $((w'_l, z'^1_l), (w''_l, z''^1_l)) \in B$. Finally, as $z''^2_l \in \|m(goto(l'_n))\|^{z''^1_l}_{e_l,w''_l}$, it follows that $((w'_h, z'_h), (w''_l, z''_l)) \in B$ by definition of B_i.

We conclude that B is an m-bisimulation between (e_h, w_h) and (e_l, w_l). Therefore, $(e_h, w_h) \sim_m (e_l, w_l)$, and therefore Σ_{goto} is a sound abstraction of Σ_{move}.

Complete abstraction: We now show that Σ_{goto} is a complete abstraction of Σ_{move}: Let $e_h, w_h \models \mathbf{K}\Sigma_{goto} \wedge \Sigma_{goto}$. We show by construction that there is a model (e_l, w_l) with $e_l, w_l \models \mathbf{K}\Sigma_{move} \wedge \Sigma_{move}$ and $(e_h, w_h) \sim_m (e_l, w_l)$. First, note that from $e_h, w_h \models \mathbf{K}\Sigma_{goto}$ it follows that $d_h(w'_h) > 0$ implies $w'_h \models \Sigma_{move}$. Now, for each w^i_h with $d_h(w^i_h) > 0$, let w^i_l be a world with $w^i_l \models \Sigma_{move}$ and such that w^i_l is like w^i_h for the high-level fluents, i.e., for every $F \notin \mathcal{F}_l$ and every $z \in \mathcal{Z}$, $w^i[F, z] = w^i_h[F, z]$. Thus, w^i is exactly like w^i_h for every fluent not mentioned in Σ_{move}. We set $e_l(w^i_l) = e_h(w^i_h)$ and $e_l(w'_l) = 0$ for every other world. Clearly, $e_l, w^i_l \models \mathbf{K}\Sigma_{move} \wedge \Sigma_{move}$. Now, let:

$$B_0 = \{((w^i_h, \langle\rangle), (w^i_l, \langle\rangle)) \mid e_h(w^i_h) > 0\}$$

As before:

$$\begin{aligned} B_{i+1} = \{((w'_h, z'_h \cdot a), (w'_l, z'_l \cdot z''_l)) \mid \\ ((w'_h, z'_h), (w'_l, z'_l)) \in B_i, \\ w'_h, z'_h \models Poss(a), z''_l \in \|m(a)\|^{z'_l}_{e_l,w_l}\} \end{aligned}$$

As each w^i_l is like w^i_h, it follows that B is definite. We can again show by induction on i that B is an m-bisimulation between (e_h, w_h) and (e_l, w_l). Therefore, $(e_h, w_h) \sim_m (e_l, w_l)$ and thus, Σ_{goto} is a complete abstraction of Σ_{move}. ◻