A History Based Approximate Epistemic Action Theory for Efficient Postdictive Reasoning

Manfred Eppe[a,*], Mehul Bhatt[b]

[a] *International Computer Science Institute, Berkeley, USA*
[b] *University of Bremen, Germany*

Abstract

We propose an approximation of the possible worlds semantics ($\mathcal{PWS}$) of knowledge with support for *postdiction* – a fundamental inference pattern for diagnostic reasoning and explanation tasks in a wide range of real-world applications such as cognitive robotics, visual perception for cognitive vision, ambient intelligence and smart environments. We present the formal framework, an operational semantics, and an analysis of soundness and completeness results therefrom.
The advantage of our approach is that only a linear number of state-variables are required to represent an agent's knowledge state. This is achieved by modeling knowledge as the history of a single approximate state, instead of using an exponential number of possible worlds like in Kripke semantics. That is, we add a temporal dimension to the knowledge representation which facilitates efficient postdiction. Since we consider knowledge **h**istories, we call our theory *h-approximation* ($\mathcal{HPX}$).
Due to the linear number of state variables, $\mathcal{HPX}$ features a comparably low computational complexity. Specifically, we show that $\mathcal{HPX}$ can solve the projection problem in polynomial (tractable) time. It can solve planning problems in NP, while e.g. for the action language $\mathcal{A}_k$[48] this is Σ_2^P-complete. In addition to the temporal dimension of knowledge, our theory supports concurrent acting and sensing, and is in this sense more expressive than existing approximations.

Keywords: commonsense reasoning, action and change, epistemic reasoning

1. Introduction

Commonsense reasoning about action & change is a vibrant research area concerned with developing formal methods for representing and reasoning about high-level knowledge [37, 7, 10, 20]. One particular line of research in reasoning about action & change is that of *epistemic* action theories, e.g. [36, 48, 47], which are concerned with reasoning about the knowledge of an agent. An agent can form knowledge directly, by executing sensing actions, or by deductive and abductive inference mechanisms.
Epistemic action theories are usually represented by a *possible-world* model of knowledge, following the work by Moore [36]. A problem with such approaches is that an exponential number of possible worlds are required to model an agent's knowledge states. Hence, possible worlds-approaches are highly intractable. Approximations with a lower complexity exist (e.g. [48]), but these usually have no support for an important form of inference, called *postdiction* – a kind of logical reasoning that is concerned with inferring knowledge about the past based on present observations (see Examples 1,2,3).
Our research addresses the development of approximate epistemic action theories that (i) natively support postdiction, (ii) have a lower complexity bound compared to approaches based on possible worlds, (iii) are provably sound, and (iv) support concurrency. This paper presents the *h-approximation* ($\mathcal{HPX}$), a history-based approximation of $\mathcal{PWS}$ that satisfies criteria (i-iv). This paper mainly elaborates on the theory of the proposed approach. An implementation of the theory in terms of answer set programming [2], along with an empirical evaluation and a case study, are presented in a follow-up paper [14].

Postdictive Reasoning

We regard postdiction as a form of reasoning accounting for causal relations between temporally ordered states. Postdiction is abductive reasoning, in the sense that it can be used to explain an observation. However, technically, it can be implemented in a deductive manner, as shown throughout this paper. Within an epistemic action theory, postdictive reasoning can be applied to verify action success and to infer new knowledge about the past. For illustration, consider the following examples:

*Corresponding author
Email addresses: eppe@icsi.berkeley.edu (Manfred Eppe), bhatt@informatik.uni-bremen.de (Mehul Bhatt)

Example 1. The litmus test
To find out whether a liquid is acidic or alkaline, one can hold a litmus paper into the liquid. If the paper is red, one can postdict that the liquid is acidic, and if the paper is blue, one knows that the liquid is alkaline.

This example, due to Moore [36], illustrates how postdiction is used to determine a world property (the acidic-ness of the liquid) by executing a non-sensing action (holding a paper in the liquid) and observing another world property (the color of the paper) that is a causal consequence of the non-sensing action.

Example 2. Grabbing an egg
Consider a household-robot with an object grabber and a vision system to identify objects. When grabbing a fragile object like an egg, the robot applies a certain force with the grabber and the egg may break. It can postdict that too much force was applied, if it observes that the egg is broken.

The example illustrates how postdiction is the underlying logical inference mechanism for learning action parameters by observing the (non-) success of the action.

Example 3. Robot passing through a door
A robot has the task to open a door and to drive through it. It can execute an open-door command, but it does not have haptic sensors to directly verify the state of the door after executing the command. Hence, considering that the door is sometimes jammed, the robot can never really know whether the door is indeed open or closed after executing the command. However, it does have a location sensor to determine on which side of the door it currently is. If the robot ends up on the other side of the door after driving, then it can postdict that the door was open, and hence that it was not jammed (assuming that the robot knows that the door was initially closed).

This example illustrates an indirect form of postdiction, where a world property (the jammed-ness of the door) is determined by executing a sequence of two non-sensing actions that are causally linked: if the door is not jammed then it will open, and if it is open then the robot will end up on the other side. The effect of the drive action (robot behind door) depends on the effect of the open-door action (open-state of door) which in turn depends on the world property that the robot will postdict (jammed-ness of the door).
In general, postdiction problems are compounded in situations where the high-level belief / reasoning module is fully independent of the low-level control framework.[1]

Contribution: $\mathcal{HPX}$ – An approximation of $\mathcal{PWS}$ with a Temporal Knowledge Dimension and Postdiction
We describe the semantics of our approximate theory by means of a transition function, where action occurrences are modeled as a transition from one knowledge state to another. Our semantics involves the following combination of key features which distinguish it from the state of the art: (i) the native capability of postdiction, (ii) an approximate state representation of linear size, (iii) the capability to represent and to reason about knowledge about the past, and (iv) concurrency of sensing and non-sensing actions. Note that all of these features are not new by themselves – it is this particular combination of features that makes our approach useful in practice, as depicted throughout this paper.

Naive and Elaboration Tolerant Postdiction
Postdiction is inherent in every diagnosis problem where the (previously unknown) reason for the success or non-success of an action is information of interest. It is therefore desireable to have action formalisms that support his kind of inference in general manner. Section 2 contains a survey that reports on different ways of implementing postdiction and identifies approximate action theories that support postdiction only in an ad-hoc manner. These action theories are often very useful due to their efficiency in terms of computational complexity, but they do not support postdiction natively and are therefore not *elaboration tolerant* [33].
According to McCarthy [33], *"[a] formalism is elaboration tolerant to the extent that it is convenient to modify a set of facts expressed in the formalism to take into account new phenomena"*. With respect to McCarthy's general characterization of elaboration tolerance, we clarify what elaboration tolerance means in the context of our work: we call an epistemic action theory elaboration tolerant, if postdiction and other knowledge-level effects of actions emerge directly from the foundational theory and do not have to be implemented manually by the domain designer. As an example consider a simple navigation problem where robots can move through doors, pick up things, etc. A potential postdictive inference in this setting is, that a door must be open if a robot successfully passed it. For an elaboration tolerant theory it should not be necessary to explicitly model rules for this postdictive inference. This is depicted in Example 4, which illustrates problems that arise when postdiction is implemented manually, by means of *static causal laws* (SCL) in the action language $\mathcal{A}_k^c$ [55].

[1] As we demonstrate in a real world case-study in [14, Sec. 4.2], this is indeed the case in typical large-scale robotics projects where multiple hybrid and mutually interacting components are involved.

Example 4. Elaboration tolerance and static causal laws
A robot can execute an action $\texttt{drive}_d$ to reach a room through a door d. A fluent $\texttt{in}$ denotes that it is in the room, and a fluent $\texttt{open}_d$ denotes that the door d is open. The robot has a location sensor to determine whether it is in the room; its activation is modeled with an action $\texttt{sense\_in}$. An epistemic action theory should be able to postdict that $\texttt{open}_d$ holds if $\texttt{in}$ holds after $\texttt{drive}_d$ and $\texttt{sense\_in}$ are executed. $\mathcal{A}_k^c$ semantics does not yield this postdictive inference natively, but one can use the following workaround:
An auxiliary fluent $\texttt{did\_drive}_d$ represents that $\texttt{drive}$ has been executed and a manually encoded static causal law (SCL) postdicts that if the robot is $\texttt{in}$ the destination room after driving the door must be $\texttt{open}$: "If $\texttt{did\_drive}_d$ and $\texttt{in}$ then $\texttt{open}_d$".
The knowledge-level effect of learning that the door must be open after passing it has to be modeled manually with the SCL, and therefore the workaround is not elaboration tolerant. This is not only circumstantial for the domain designer, but also dangerous, as the approach can easily generate wrong beliefs. For example, consider a scene with two doors that lead to the same room, and static causal laws for both doors, i.e. $d \in \{1, 2\}$. Consider further, that the robot executes first $\texttt{drive}_1$, but the door is closed. Then it executes $\texttt{drive}_2$ and senses its location ($\texttt{sense\_in}$). If door 2 is open and $\texttt{in}$ is known to hold, then the SCL for door 1 will fire and generate the wrong belief that door 1 is open.

Linear Number of State Variables – Tractable Projection Problem
Epistemic action theories that are based on a possible-world semantics ($\mathcal{PWS}$) have the problem that an exponential number of possible worlds are required to model an agent's knowledge state. Given that $|F|$ is the number of potentially unknown fluents of a domain specification, $2^{|F|} \cdot |F|$ variables are required to model the knowledge state of the agent.
Our approach requires only a linear number of state variables; given that t is the number of state transitions (or steps), $\mathcal{HPX}$ requires $|F| \cdot (t+1)$ state variables to model an agent's knowledge state. Section 4 shows that for this reason solving the *projection problem* remains polynomial, while postdiction is possible.

Temporal Knowledge Dimension
Native and elaboration tolerant support for postdiction is achieved by incorporating a temporal knowledge dimension. For instance, if an agent moves trough a door (say at $t = 2$) and later (say at $t = 4$) comes to know that it is behind the door, then it can postdict that the door must have been open at $t = 2$. After applying this postdictive inference it can further refine its knowledge and use the inertia assumption to infer that the door is still open at $t = 3$ and $t = 4$.

Concurrent Sensing and Action
The temporal dimension of knowledge also allows one to elegantly model concurrent execution of sensing and physical actions. This is non-trivial, especially if an action senses the value of a fluent while concurrently changing it, as depicted with the following extended version of the Yale shooting problem [19].

Example 5. The Yale shooting problem with concurrent acting and sensing
Consider an agent shooting at a turkey. It can sense whether the gun was loaded by hearing the explosion's noise, and assuming that there was an explosion, the agent can conclude that the turkey must be dead after the shooting. Note that this version of the problem requires us to model the shooting action with concurrent sensing (gun loaded if explosion heard) and physical effects (turkey dead, gun unloaded).

Modeling this scenario correctly, such that a conclusion about the turkey's death follows, is not possible with existing approaches, except [31] (for details see Example 9).

2. Epistemic Action Theory: A Survey

Our survey distinguishes four types of epistemic action theories (TH-1 — TH-4), as summarized in Table 1.

TH-1 *Theories based on a $\mathcal{PWS}$.* This is the traditional approach, following the work by [21, 26, 36]. Model-theoretic $\mathcal{PWS}$-based approaches (e.g. [47]) typically employ Kripke's accessibility relation to model knowledge. Operational approaches use multisets of fluents (e.g. [48], [30], [12]).

TH-2 *Theories based on disjunctive state-representations* (e.g. [40]). Their expressiveness and inference capabilities are comparable to $\mathcal{PWS}$ approaches, but corresponding implementations can be more efficient in practice [53].

TH-3 *Approximations of $\mathcal{PWS}$* (e.g. [48]). These are less expressive but have better computational complexity properties than $\mathcal{PWS}$. Such theories are equivalent to $\mathcal{PWS}$-based approaches, at the cost that strong restrictions are imposed [43, 6].

Features of epistemic action theories	# State variables	Elaboration tolerant postdiction	Temporal knowledge dimension	Concurrent acting and sensing	Implementation	Soundness / completeness results
TH-1 – $\mathcal{PWS}$-based approaches						
Extension to $\mathcal{A}$ Lobo et al. [30]	EXP	✓	-	-	-	✓
$\mathcal{A}_k(\mathcal{PWS})$ Son & Baral [48]	EXP	✓	-	-	-	✓
Epistemic SC Scherl & Levesque [47]	EXP	✓	-	-	-	✓
$\mathcal{AOL}$ Lakemeyer & Levesque [27]	∞	✓	-	-	-	-
Epistemic FC Thielscher [50]	EXP	✓	-	-	(✓)	✓
CFF Hoffmann & Brafman [22]	EXP	✓	-	-	✓	-
EFEC Ma et al. [31], Miller et al. [35]	EXP	✓	✓	✓	✓	-
Approach by Vlaeminck et al. [56]	EXP	✓	✓	-	-	-
Approach by Fagin et al. [16]	EXP	✓	✓	-	-	✓
TH-2 – Disjunctive state approaches						
DECKT Patkos & Plexousakis [40]	EXP	✓	-	(✓)	✓	✓
PrAO To [54]	EXP	✓	-	-	✓	-
TH-3 – Approximations of $\mathcal{PWS}$						
h-approximation ($\mathcal{HPX}$)	**LIN**	✓	✓	✓	✓	✓
Epistemic SC Demolombe & del Pilar Pozos Parra [11]	LIN	-	-	-	-	-
Epistemic SC Liu & Levesque [29]	LIN	-	-	-	-	-
$\mathcal{A}_k^c$(0-approximation) Tu et al. [55]	LIN	-	-	(✓)	✓	✓
TH-4 – Approaches requiring explicit knowledge-level effect specification						
PKS Petrick & Bacchus [42]	LIN	-	-	(✓)	✓	-
FLUX Thielscher [52]	LIN	-	-	(✓)	✓	(✓)
INDIGOLOG de Giacomo & Levesque [18]	LIN	-	-	(✓)	✓	-

Table 1: Survey of epistemic action theories

TH-4 *Theories that rely on explicit formulation of knowledge-level effects* (e.g. [42]). Theories of this category can be understood as approximations too, but it is required to carefully specify knowledge-level effects of actions in an epistemically accurate manner. For instance, one has to explicitly model, that if the condition of an action is unknown, then its effect is unknown.

Table 1 identifies existing approaches and compares their computational properties, expressiveness and inference capabilities. We are particularly interested in the following features:

1. *Number of state variables (exponential or linear) and computational complexity.*
   The number of state variables has a tremendous influence on the computational complexity of action formalisms. For instance, consider the action language $\mathcal{A}_k$ [48]: the plan existence problem is Σ_2^P-complete for an exponential number of state variables and NP-complete for the 0-approximation where the number of state variables is linear wrt. the domain size [3].[2] For the epistemic Situation Calculus [47] we have an exponential number of state variables, which gives an exponential number of possible worlds even if we restrict the variables to be boolean. $\mathcal{AOL}$ [27] has an infinite number of state variables, because it allows for nested knowledge.
2. *Elaboration tolerant postdiction.*
   Our work is motivated by the need for postdiction to solve diagnosis and explanation problems in epistemic action theory. Example 4 illustrates the need for *elaboration toleranance* [33] within a formalism for postdiction. That is, we are interested in whether knowledge-level effects of actions emerge from the underlying foundational theory of a formalism, or if they have to be implemented manually by the domain designer.
3. *Temporal knowledge dimension.*
   Reasoning about past (or future) facts opens up a new range of problems. For instance, in narrative interpretation or forensic reasoning. Witnesses may give evidence about facts in the past; or knowledge about the occurrence of an event may be acquired at a later time point.
4. *Concurrent acting and sensing.*
   Real-world domains often demand to model actions which change the world and concurrently sense a property of the world.

[2] Under the assumption that plans are polynomial in size and the number of sensing actions is limited.

For instance, pulling the trigger of a gun causes the shooter to know whether the gun was loaded and at the same time has the physical effect of the impact of the bullet (see Example 9). Another more practical example is a sensor that consumes energy while the sensing happens.

5. *Implementation.*
   A major goal of our research is to use $\mathcal{HPX}$ with real robots and other applications that require an implementation. An implementation also serves as a proof-of concept of the formalism. We present an answer set programming [2] implementation of $\mathcal{HPX}$ in [14].
6. *Soundness / completeness results.*
   Many formalism are defined in 'isolation' wrt. other approaches, i.e. the relation to other formalisms is not investigated formally by performing soundness or completeness proofs. Without such proofs, the relation between different action theories is unclear and conditions under which a formalism is equally expressive as another formalism can not be identified.

We admit that this list of features emphasises the capabilities of the $\mathcal{HPX}$ formalism, but argue that in particular postdiction and a low computational complexity are, combined, of general importance in real world problems. Of course, the invesitgated other formalisms have other special strengths, which we discuss in the following.

TH-1. *$\mathcal{PWS}$-based Epistemic Action Theories*

The most popular approach to represent knowledge in an action theoretic context is a non-approximated $\mathcal{PWS}$, where knowledge is represented by an exponential number of possible worlds. The approach was first applied by Moore [36], and stems from the work concerning Epistemic Modal Logic by Carnap [8], Hintikka [21], Kripke [26] and others. Here, knowledge states are modeled either with an *accessibility relation* (e.g. [47]) between possible worlds, or with multisets of fluents in mathematical logic (e.g. [48]). These approaches support postdictive reasoning in an elaboration tolerant manner, but they have the disadvantage that modeling an agent's knowledge state requires an exponential number of possible worlds and hence an exponential number of state variables.
As an example for such an approach consider Lobo et al. [30], who used both mathematical logic and epistemic logic programming to formulate a $\mathcal{PWS}$ based epistemic extension to the action language $\mathcal{A}$. Another $\mathcal{PWS}$ based epistemic semantics for $\mathcal{A}$ is defined for the action language $\mathcal{A}_k$ by Son & Baral [48]. $\mathcal{A}_k$ is sound and complete wrt. the approach by Scherl & Levesque [47] and the approach by [30]. However, we could not locate a full implementation, and $\mathcal{A}_k$ does not feature concurrent actions in general.
Scherl & Levesque [47] provide an epistemic extension and a solution to the frame problem for the Situation Calculus [32] using an accessibility relation. A temporal knowledge dimension does not exist and concurrency is not supported. To the best of our knowledge there exists no implementation of their version of the epistemic SC. Son & Baral [48] showed that under certain assumptions the approach by Scherl & Levesque [47] is sound and complete wrt. $\mathcal{A}_k$.
Lakemeyer & Levesque [27] combine knowledge and action in the logic $\mathcal{AOL}$ in the context of the situation calculus and the logic of *only knowing* [28]. Their approach considers introspection, i.e. knowledge about knowledge, and for this reason the number of state variables is in infinite. A temporal dimension of knowledge and concurrency is not considered. To the best of our knowledge, the theory has not been implemented and the relation to other epistemic action calculi has not been investigated formally.
The Fluent Calculus (FC) [23, 49] was extended to consider knowledge in [50]. It uses an accessibility relation similar to [47] and hence requires an exponential number of state variables. The epistemic FC does not consider a temporal knowledge dimension. There exists an extension which covers concurrency [51], but this extension does not consider knowledge and sensing. Kahramanogullari & Thielscher [24] provide a formal investigation concerning the relation between FC and the epistemic extension of $\mathcal{A}$ by Lobo et al. [30]. There exists a Prolog implementation of the epistemic Fluent Calculus called FLUX [52], but its semantics differs from the original epistemic FC in that it does not employ an accessibility relation. For this reason we report about FLUX separately.
In addition to logic-based action-theoretic approaches there exist several PDDL-based [34] planners that deal with incomplete knowledge. These planners achieve high performance via practical optimizations such as BDDs [4] or heuristics-driven search algorithms like those used for the ContingentFF (CFF) planner [22]. However, their underlying semantics is typically based on a $\mathcal{PWS}$. For instance, the semantics of CFF is based on sequential STRIPS [17] and adds the $\mathcal{PWS}$-approach to model sensing and knowledge. Despite its efficiency in relatively small domains, the exponential number of state variables causes an extreme phase transition if the domain size exceeds a certain threshold.[3] Postdiction is possible and elaboration tolerant, but concurrent acting and sensing is not supported in CFF. A formal investigation of the relation between CFF and other formalisms is also not provided.
Concerning the temporal dimension of knowledge two approaches stand out, namely Ma et al. [31] and Vlaeminck et al. [56]. Ma et al. [31] propose an epistemic extension to the Event Calculus [25] which considers *possible world histories* instead of possible

[3] For example, the RING problem [22] where an agent must move through n rooms and close the windows in these rooms demands 1.5s for 4 rooms, 480s for 5 rooms and produces a timeout $>$ 3600s for 6 rooms with CFF on a 2 Ghz i5 machine with 6GB RAM.

worlds. It has elaboration tolerant support for postdiction and knowledge about past and future can be represented. Concurrent acting and sensing is possible with EFEC, and an implementation exists as answer set programming.[4]
Another $\mathcal{PWS}$-based first-order logical framework to model a temporal dimension of knowledge is provided by Vlaeminck et al. [56]. The framework employs first-order-reasoning such that it allows for elaboration tolerant postdiction and the projection problem is solvable in polynomial time. However, the authors do not provide a practical implementation and evaluation of their method. The key feature of their approach is that the first-order reasoning is approximated, such that the plan-existence problem is in NP despite the exponential number of state variables of the actual action formalism.[5] The framework has not been implemented and the relation to other action calculi has not been formally investigated.
Fagin et al. [16] present a very comprehensive earlier approach to formalise the reasoning about knowledge in action. The authors give an explicit account on time, even in the multi agent case. However, the work is rather theoretical and therefore hardly comparable with our work which is geared towards efficiency and the concrete application in robotics.

TH-2. *Theories with a Disjunctive Knowledge State Representation*

There are alternatives to a $\mathcal{PWS}$-based knowledge representations which use a disjunctive knowledge representation. One example is DECKT [40] – an epistemic extension to the Event Calculus. DECKT relies on so-called *Hidden Causal Dependencies* which are modeled by reified first-order predicates. For instance $Knows(\neg f_1 \vee f_2, T)$ represents that at a time T an agent has knowledge about the causal dependency "if f_1 holds then f_2 must also hold". In general, disjunctive approaches require only a linear number of states to model an agent's knowledge but the number of variables per state is up to exponential. An implementation of DECKT is presented in [41] but the implementation is not capable of action planning. DECKT is sound and complete wrt. BDECKT [40], a $\mathcal{PWS}$-based version of the Event Calculus. Concurrent acting and sensing is in principle possible but fails if an action senses the value of a fluent which is modified at the same time, as demonstrated in our Yale Shooting Scenario in Example 9.
Another approach by To [54] implements a general planning framework called PrAO which is based on so-called *minimalDNS* representations. The approach is reported to have a very good performance compared to $\mathcal{PWS}$-based planners, but concurrency is not supported and a formal comparison with other epistemic action calculi is not presented.

TH-3. *Approximate Epistemic Action Theories*

Approximate theories are usually derived from a $\mathcal{PWS}$ based formalization. Approximations can have simpler state representations and hence a lower computational complexity, but postdiction is not naively supported with existing approaches. Demolombe & del Pilar Pozos Parra [11] provide an approximate epistemic extension to the Situation Calculus (SC) which is based on special knowledge fluents. Their approach involves simpler frame axioms compared to the $\mathcal{PWS}$ based approach by Scherl & Levesque [47], such that an implementation would be tractable. However, postdiction is not possible with this approach. A temporal knowledge dimension and concurrency is also not supported. Liu & Levesque [29] present another epistemic extension of the Situation calculus. Their approach to approximate $\mathcal{PWS}$ is based on so-called *local action effects*. The intuition behind local action effects is that all conditions of an action must be known before execution. This implies that postdiction – in the sense of inferring the conditions of an action by observing its effect – is not required. The approach by [29] is claimed to coincide with the 0-approximation of Son & Baral [48] if formulae are restricted to be propositional. However, we were unable to locate a formal proof.
The *0-approximation* by Son & Baral [48] does not consider an exponential number of possible worlds, but instead one single approximate world which is the intersection of all possible worlds. This requires only a linear number of state variables to model the knowledge state of an agent, instead of an exponential number with $\mathcal{PWS}$ based approaches. The plan-existence problem for $\mathcal{A}_k$ is NP-complete [3], and postdiction is not supported.[6] Tu et al. [55] introduce $\mathcal{A}_k^c$ and add Static Causal Laws (SCLs) to the 0-approximated $\mathcal{A}_k$. In Example 4 we demonstrate that SCL allow only for a non-elaboration tolerant form of Postdiction.

TH-4. *Epistemic Action Theories with Explicit Knowledge-Level Effects*

Approaches like the PKS planner [42], the FLUX system [52] or INDIGOLOG [18] require one to model the knowledge-level effects of an agent explicitly in the action specification. These are able to deal with incomplete knowledge, but knowledge-level effects of actions have to be defined manually for each action. This leads to elaboration intolerance and allows a domain designer to specify epistemically inaccurate domain specifications. For example, a domain designer has to implement manually that the effect literal of an action becomes unknown if the condition literal is unknown. We argue that such knowledge-level effects should be handled by the theory, not by the domain designer.
PKS is based on the Epistemic Situation Calculus by [47], but a formal soundness proof wrt. this or other formalisms is not presented. There exists an implementation of PKS which is particularly efficient if many functional fluents are used. FLUX [52] can be

[4] An implementation can be found at http://www.ucl.ac.uk/infostudies/efec/ (accessed on 17th July 2014)
[5] The authors only provide a complexity result that the projection problem is polynomial, but it follows directly from the definition of non-deterministic Turing Machines that plan-existence is in NP.
[6] Son & Baral [48] present other approximations (e.g. 1-approximation or ω-approximation) in the same paper as well, but these also do not support postdiction.

understood as an implementation of the (epistemic) Fluent Calculus in Prolog. However, there is a fundamental difference between the epistemological reasoning machineries of both calculi: Unlike the original epistemic Fluent Calculus defined in [50], FLUX does not employ an accessibility relation but instead an explicit knowledge-predicate. As a result, knowledge-level effects of actions (such as postdiction) have to be specified manually which makes a formal investigation of the relation between this limited form of epistemic reasoning and the epistemic FC difficult; soundness and completeness results are not provided. The manual specification of knowledge-level effects make FLUX also elaboration-intolerant. INDIGOLOG [18] is a high-level cognitive robotics control framework that supports sensing and incomplete knowledge. It is based on an operational semantics and implemented in Prolog. In the implementation, sensing is modeled in a simplified manner and an accessibility relation is not employed. Though work concerning epistemic accuracy of INDOGOLOG has been conducted in [46], we were unable to find actual soundness or completeness results for INDIGOLOG's semantics wrt. other epistemic action theories such as [47].
Concurrent acting and sensing is possible with PKS, FLUX and INDIGOLOG but it fails for actions which sense and concurrently change the same fluent's value, as described in our extended version of the Yale Shooting Problem (Example 9).

3. The h-Approximation – An Approximate Semantics for Postdictive Reasoning

This section describes the operational semantics of the h-approximation. In Section 3.1 we describe the syntax of the action language and show how this maps to a set-theoretic formalization of $\mathcal{HPX}$. The formalization is based on so-called *h-states* which represent the knowledge state of an agent. An h-state consists of a set of pairs of fluent literals and time points that represent the *knowledge history* of an agent, as well as a set of pairs of actions and time points to represent the *action history*.
Section 3.2 describes how state transitions are modeled. We define a transition function which maps an h-state and a set of actions to a set of h-states. State transitions involve a re-evaluation step which iteratively refines the knowledge history of an agent by considering sensing results and the occurrence of actions. Based on the basic transition function for single actions, we formalize concurrent conditional plans (CCP) and define an extended transition function which maps a CCP and an h-state to a set of h-states. We illustrate the theory with a running minimal example of a robot trying to enter a room by driving through a door.

3.1. Domain Specification and Syntax

Throughout this paper we use the following notational conventions: We use the symbol a to denote *actions*, ep for *effect proposition*, n and t for *time* (or step), b for *branch*, and f for *fluent*. l denotes *fluent literals* of the form f or $\neg f$. $\bar{l}$ denotes the complement of l and $|l|$ is used to "positify" a literal, i.e. $|\neg f| = f$ and $|f| = f$. With these conventions we describe the syntax of language to specify domains denoted $\mathcal{D}$. Domain specifications are quadruples $\langle \mathcal{VP}, \mathcal{EP}, \mathcal{KP}, \mathcal{EXC} \rangle$ that are defined by action language elements (1a) – (1d).

$$\textbf{initially}(l_1^{init}, \ldots, l_{n_{init}}^{init}) \tag{1a}$$

$$\textbf{causes}\big(a, l^e, l_1^c \ldots l_{n_c}^c\big) \tag{1b}$$

$$\textbf{determines}(a, f^s) \tag{1c}$$

$$\textbf{executable}\big(a, l_1^{ex} \ldots l_{n_{ex}}^{ex}\big) \tag{1d}$$

(1a). A set of *value propositions* (VPs) denotes initial knowledge. We define $\mathcal{VP}$ as the set of expressions (1a). We sometimes write $\textbf{initially}\big(l_1^{init}, \ldots, l_{n_{init}}^{init}\big)$ to denote a set of n_{init} VPs for the respective literals $l_1^{init}, \ldots, l_{n_{init}}^{init}$.

(1b). A set of *effect propositions* (EPs) represent conditional action effects . We call $l_1^c \ldots l_{n_c}^c$ condition literals and l^e an effect literal. For an effect proposition ep of the form (1b) we write $c(ep)$ to denote the set of condition literals and $e(ep)$ for the effect literal. We use $\mathcal{EP}$ to denote the set of effect propositions of a domain $\mathcal{D}$, and we sometimes write $\mathcal{EP}^a$ to denote the subset of EPs for one fixed a.

(1c). *Knowledge propositions* (KPs) of the form (1c) represent that an action a senses a fluent f^s. We define $\mathcal{KP}$ as the set of knowledge propositions of $\mathcal{D}$, and we sometimes write $\mathcal{KP}^a$ to denote the knowledge proposition for one fixed a.

(1d). *Executability conditions* (EXCs) of the form (1d) denote what an agent must know in order to execute an action. We denote $\mathcal{EXC}$ as the set of executability conditions of $\mathcal{D}$, and we sometimes write $\mathcal{EXC}^a$ to denote the set of executability conditions for one fixed a. Formally, each action has one executabiltiy condition, where $\textbf{executable}(a, \emptyset)$ is implicit for any action a without an explicit executability condition.

Note that a set of domain fluents (denoted $\mathcal{F}_\mathcal{D}$) is implicitly defined by the domain definition: Whenever a fluent literal f or $\neg f$ is involved in one of the language elements (1a) – (1d), then f is contained in the set of domain-fluents $\mathcal{F}_\mathcal{D}$. Respectively, $\mathcal{L}_\mathcal{D}$ is the set of domain literals.

3.2. Operational Semantics of the h-Approximation

The $\mathcal{HPX}$ semantics is based on so-called *history-states* (h-states) and a transition function (6). The transition function maps actions and h-states to h-states. To realize postdiction and other epistemic effects, a re-evaluation function $eval$ is applied after each state transition. $eval$ retrospectively considers temporal knowledge and incrementally refines knowledge with inference mechanisms for postdiction, causation and inertia. In addition to the transition function (6) for single actions we define an extended transition function (16) that maps a *concurrent conditional plan* and a state to a set of states.[7]

3.2.1. Knowledge States with a Temporal Dimension

Formally, h-states $\mathfrak{h}$ are pairs $\langle \boldsymbol{\alpha}, \boldsymbol{\kappa} \rangle$, where $\boldsymbol{\alpha}$ denotes the action history and $\boldsymbol{\kappa}$ the knowledge history of $\mathfrak{h}$. Formally,

- an *action history* $\boldsymbol{\alpha}$ consists of pairs $\langle a, t \rangle$ where a is an action and t is a time step, and
- a *knowledge history* $\boldsymbol{\kappa}$ is a set of pairs $\langle l, t \rangle$ where l is a literal and t is a time-step.

To handle *concurrency* in a more convenient manner we introduce *effect histories* as an auxiliary instrument derived from action histories.

- An *effect history* $\boldsymbol{\epsilon}$ is a set of pairs $\langle ep, t \rangle$ where ep is an effect proposition and t is a time-step.

The formal definition of effect histories is provided in Definition 1.

Definition 1 (Effect history $\boldsymbol{\epsilon}$). *Let $\boldsymbol{\alpha} = \{\langle a_1, t_1 \rangle, \ldots, \langle a_n, t_n \rangle\}$ be an action history and let $\mathcal{EP}^a$ denote the set of effect propositions of a domain $\mathcal{D}$ of the form (1b) for a fixed a. Then the effect history $\boldsymbol{\epsilon}(\boldsymbol{\alpha})$ of the action history $\boldsymbol{\alpha}$ is given by (2).*

$$\boldsymbol{\epsilon}(\boldsymbol{\alpha}) = \{\langle ep, t \rangle \,|\exists\, \langle a, t \rangle \in \boldsymbol{\alpha} : ep \in \mathcal{EP}^a\} \tag{2}$$

For convenience we also write (3).

$$\boldsymbol{\epsilon}(\mathfrak{h}) = \boldsymbol{\epsilon}(\boldsymbol{\alpha}(\mathfrak{h})) \tag{3}$$

In general, we write $\boldsymbol{\alpha}(\mathfrak{h})$, $\boldsymbol{\kappa}(\mathfrak{h})$ and $\boldsymbol{\epsilon}(\mathfrak{h})$ to denote the action history, knowledge history and effect history of an h-state $\mathfrak{h}$. To simplify notation, we sometimes transfer sub- and superscripts from $\mathfrak{h}$ to $\boldsymbol{\epsilon}$, $\boldsymbol{\kappa}$ and $\boldsymbol{\alpha}$ (if clear from the context). For instance we write $\boldsymbol{\epsilon}_n$ to denote $\boldsymbol{\epsilon}(\mathfrak{h}_n)$.

3.2.2. Initial Knowledge

A particular h-state of a domain description is the *initial state*, described by Definition 2.

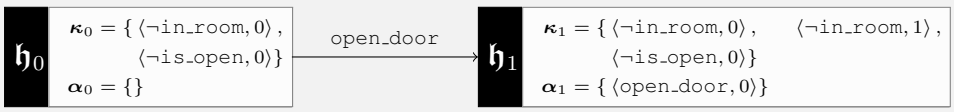
Definition 2 (Initial h-state $\mathfrak{h}_0$). *A state is called the* initial state *(denoted by $\mathfrak{h}_0 = \langle \boldsymbol{\alpha}_0, \boldsymbol{\kappa}_0 \rangle$) of a domain $\mathcal{D}$ if and only if $\boldsymbol{\alpha}_0 = \emptyset$, and for every fluent literal l in a value proposition $\mathcal{VP}$: $\langle l, 0 \rangle \in \boldsymbol{\kappa}_0$.*

Example 6 depicts how an action is applied to transform the initial state $\mathfrak{h}_0$ into a successor state $\mathfrak{h}_1$.

Example 6. Action and knowledge history
Consider domain $\mathcal{D}_1$ which specifies the problem of opening a door into a room if it is unknown whether the door is jammed.

$$\mathcal{D}_1 = \left\{ \begin{array}{r} \textbf{initially}\,(\neg\texttt{in\_room}, \neg\texttt{is\_open}) \\ \textbf{causes}\,(\texttt{open\_door}, \texttt{is\_open}, \neg\texttt{jammed}) \end{array} \right\} \tag{$\mathcal{D}_1$}$$

The value proposition results in the initial knowledge history $\boldsymbol{\kappa}_0 = \{\langle \neg\texttt{in\_room}, 0 \rangle, \langle \neg\texttt{is\_open}, 0 \rangle\}$, and we have the action history $\boldsymbol{\alpha}_0 = \emptyset$. Applying action `open_door` on $\mathfrak{h}_0$ causes a transition to $\mathfrak{h}_1$ as follows:



Note that $\boldsymbol{\kappa}_1$ does not contain a pair $\langle \texttt{in\_open}, 1 \rangle$ or $\langle \neg\texttt{is\_open}, 1 \rangle$: If it is unknown whether there is an abnormality in opening the door, then it is also unknown whether the door is actually open after executing this action.

[7] Our definition of concurrency is restricted in the sense that all actions are assumed to have the same duration. Therefore we define concurrency only wrt. single actions and not wrt. to other concurrent conditional (sub-)plans.

3.2.3. Knowledge about the Present (and the Past)

To identify the *present world state* within a knowledge history we define an auxiliary function now (4), which it returns the number of state transitions that have occurred so far.

$$now(\mathfrak{h}) = \begin{cases} 0 & \text{if } \boldsymbol{\alpha}(\mathfrak{h}) = \emptyset \\ t+1 & \text{otherwise, where } t \text{ is maximal in all tuples } \langle a, t\rangle \in \boldsymbol{\alpha}(\mathfrak{h}) \end{cases} \tag{4}$$

To query the knowledge of an agent that is in an epistemic state $\mathfrak{h}$, we use an entailment operator $\models$ to define (i) whether at the present step a literal l is known to hold *at the present step* (5a), or (ii) whether at the present step a pair $\langle l, t\rangle$ is known to hold (5b), i.e. whether l is known to hold *at a possibly earlier step* $t \leq now(\mathfrak{h})$.

$$\mathfrak{h} \models l \Leftrightarrow \langle l, now(\mathfrak{h})\rangle \in \boldsymbol{\kappa}(\mathfrak{h}) \tag{5a}$$

$$\mathfrak{h} \models \langle l, t\rangle \Leftrightarrow \langle l, t\rangle \in \boldsymbol{\kappa}(\mathfrak{h}) \tag{5b}$$

3.2.4. Executability of Actions

Executability conditions (1d) are qualifications on the agent's knowledge at the time it executes an action. They reflect what an agent must know in order to execute an action. Executability conditions are formalized in Definition 3.

Definition 3 (Executability of actions). *Consider an action a with an executability condition* ***executable***$\left(a, l_1^{ex} \ldots l_{n_{ex}}^{ex}\right)$*, i.e. ,* $\mathcal{EXC}^a = \{l_1^{ex}, \ldots, l_{n_{ex}}^{ex}\}$*. We say that a is executable in an h-state $\mathfrak{h}$ if* $\forall l^{ex} \in \mathcal{EXC}^a : \mathfrak{h} \models \langle l^{ex}, now(\mathfrak{h})\rangle$*.*
We say that an h-state $\mathfrak{h} = \langle \boldsymbol{\alpha}, \boldsymbol{\kappa}\rangle$ is execute-consistent*, if all past and present actions are executable, i.e.,* $\forall \langle a, t\rangle \in \boldsymbol{\alpha}, \forall l^{ex} \in \mathcal{EXC}^a : \boldsymbol{\kappa} \models \langle l, t\rangle$*.*

Intuitively, an action is executable if all literals in the executability condition are known to hold at the step the action is executed, i.e. at $now(\mathfrak{h})$.

3.2.5. Sensing, Branching, Transition Function

The transition function Ψ (6) adds a set of actions to the action history $\boldsymbol{\alpha}$ and then evaluates the knowledge-level effects of these actions. Ψ considers sensing and maps a set of actions $\boldsymbol{A}$ and a state to a set of states.

$$\begin{aligned} \Psi(\mathbf{A}, \mathfrak{h}) = & \bigcup_{k \in sense(\mathbf{A}^{ex}, \mathfrak{h})} eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(\mathfrak{h}) \cup k\rangle) \\ \text{where } & \bullet\ \boldsymbol{A}^{ex} \text{ is the subset of actions of } \boldsymbol{A} \text{ which are executable in } \mathfrak{h} \\ & \bullet\ \boldsymbol{\alpha}' = \boldsymbol{\alpha}(\mathfrak{h}) \cup \{\langle a, t\rangle \,|a \in \mathbf{A}^{ex} \wedge t = now(\mathfrak{h})\} \end{aligned} \tag{6}$$

The transition function involves two other functions, namely $sense$ and $eval$:

- $eval$ (15) is a re-evaluation function which we describe in Section 3.2.7. In brief, $eval$ refines the knowledge-history of an h-state by determining the knowledge-level effects of non-sensing actions using certain inference mechanisms.
- $sense$ (7) adds sensing results to the knowledge history. It is formally defined as follows:

$$sense(\mathbf{A}, \mathfrak{h}) = \begin{cases} \{\{\langle f^s, t^s\rangle\}, \{\langle \neg f^s, t^s\rangle\}\} & \text{if } \mathbf{A} \text{ contains exactly one sensing action } a \text{ with a knowledge proposition} \\ & \textbf{determines}(a, f^s) \text{ and } \{\langle f^s, t^s\rangle, \langle \neg f^s, t^s\rangle\} \cap \boldsymbol{\kappa}(\mathfrak{h}) = \emptyset \\ \{\emptyset\} & \text{otherwise} \end{cases}$$
$$\text{where } t^s = now(\mathfrak{h}) \tag{7}$$

Intuitively, $sense$ describes that knowledge is added to the original h-state if none of the possible outcomes of the sensing (either f^s or $\neg f^s$) is already known. Note that the time at which the sensing result holds is the time at which the sensing happens, i.e. the time before the successor-state time: $t^s = now(\mathfrak{h})$. Example 9 in Appendix A.1 demonstrates that this is essential to model concurrent acting and sensing. We restrict sensing to only one fluent at a time, because sensing more that one fluent at once would lead to a state space explosion caused by an exponential branching factor.
The re-evaluation function $eval$ consists of the following five inference mechanisms which constitute the re-evaluation process.

*3.2.6. Inference Mechanisms (**IM1.–IM.5**)*

The inference mechanisms **IM.1 – IM.5** are crucial for the postictive reasoning and temporal knowledge dimension. For a thorough illustration we refer the reader to Examples 10 – 12 in Appendix A.2.

► IM.1 – IM.2: Inertia

To define how knowledge persists, we first formalize inertia (8). Intuitively, a literal l is inertial at a step t if no effect proposition can negate l.

$$inertial(l,t,\mathfrak{h}) = \begin{cases} true & \text{if } \forall \langle ep,t\rangle \in \boldsymbol{\epsilon}(\mathfrak{h}) : \big(e(ep) = \bar{l}\big) \Rightarrow \big(\exists l^c \in c(ep) : \langle \overline{l^c}, t\rangle \in \boldsymbol{\kappa}(\mathfrak{h})\big) \\ false & \text{otherwise} \end{cases} \tag{8}$$

A literal l is inertial at a step t if (i) there is no effect proposition (EP) such that $\langle ep,t\rangle \in \boldsymbol{\epsilon}(\mathfrak{h})$ and ep has a complementary effect literal $\bar{l}$, or (ii) there is an EP such that $\langle ep,t\rangle \in \boldsymbol{\epsilon}(\mathfrak{h})$ and ep has a complementary effect literal $\bar{l}$, but ep has at least one condition literal l^c which is known not to hold at t.

Having defined when a fluent is inertial, we can define forward and backward inertia of knowledge. To this end, we state two functions fwd (9) and $back$ (10) that map an h-state to an h-state. Forward inertia is defined by (9).

$$\begin{aligned} &fwd(\mathfrak{h}) = \langle \boldsymbol{\alpha}(\mathfrak{h}), \boldsymbol{\kappa}(\mathfrak{h}) \cup add_{fwd}(\mathfrak{h})\rangle \\ &\text{where} \\ &add_{fwd}(\mathfrak{h}) = \{\langle l,t\rangle \mid \langle l,t-1\rangle \in \boldsymbol{\kappa}(\mathfrak{h}) \wedge inertial(l,t-1,\mathfrak{h}) \wedge t \leq now(\mathfrak{h})\} \end{aligned} \tag{9}$$

Backward inertia is defined by (10).

$$\begin{aligned} &back(\mathfrak{h}) = \langle \boldsymbol{\alpha}(\mathfrak{h}), \boldsymbol{\kappa}(\mathfrak{h}) \cup add_{back}(\mathfrak{h})\rangle \\ &\text{where} \\ &add_{back}(\mathfrak{h}) = \{\langle l,t\rangle \mid \langle l,t+1\rangle \in \boldsymbol{\kappa}(\mathfrak{h}) \wedge inertial(\bar{l},t,\mathfrak{h}) \wedge t \geq 0\} \end{aligned} \tag{10}$$

For example, consider that at a step $t = 2$ the open-state of a door is unknown. An action `sense_open` with a knowledge proposition **determines** (`sense_open`, `is_open`) would produce two sensing results, $\langle$`is_open`$, 2\rangle$ and $\langle\neg$`is_open`$, 2\rangle$. Let us consider the positive case, where $\langle$`is_open`$, 2\rangle$ is produced. Given that the fluent `is_open` is inertial at step 2, forward inertia (9) will produce a pair $\langle$`is_open`$, 3\rangle$, and, given that $\neg$`is_open` is inertial at step 1, backward inertia (10) will produce $\langle\neg$`is_open`$, 1\rangle$. For a detailed illustration of (9) and (10) consider the steps from $\tilde{\mathfrak{h}}_2^{1^+}$ to $\tilde{\mathfrak{h}}_2^{2^+}$ and from $\tilde{\mathfrak{h}}_2^{2^+}$ to $\tilde{\mathfrak{h}}_2^{3^+}$ in Example 10, Appendix A.2.

► IM.3: Causation

We define a function $cause$ that produces knowledge about the effects of executable actions if the conditions are known. Intuitively, if an effect proposition ep is applied at $t-1$, and all condition literals of ep are known to hold at $t-1$, then the effect literal l^e of ep holds at step t.

$$\begin{aligned} &cause(\mathfrak{h}) = \langle \boldsymbol{\alpha}(\mathfrak{h}), \boldsymbol{\kappa}(\mathfrak{h}) \cup add_{cause}(\mathfrak{h})\rangle \\ &\text{where} \\ &add_{cause}(\mathfrak{h}) = \{\langle l^e,t\rangle \mid \exists \langle ep,t-1\rangle \in \boldsymbol{\epsilon}(\mathfrak{h}) : \{\langle l_1^c,t-1\rangle, \ldots, \langle l_n^c,t-1\rangle\} \subseteq \boldsymbol{\kappa}(\mathfrak{h})\} \\ &\text{with } c(ep) = \{l_1^c, \ldots, l_k^c\} \text{ and } e(ep) = l^e. \end{aligned} \tag{11}$$

► IM.4 – IM.5: Positive and negative postdiction

The function pd^{pos} (12) defines *positive postdiction*. This inference shows that the conditions of an effect proposition hold if *(i)* the effect literal is known to hold after a set of concurrent actions are executed and *(ii)* known not to hold before action execution and *(iii)* no other effect proposition could have triggered the effect literal.

$$\begin{aligned} &pd^{pos}(\mathfrak{h}) = \langle \boldsymbol{\alpha}(\mathfrak{h}), \boldsymbol{\kappa}(\mathfrak{h}) \cup add_{pd^{pos}}(\mathfrak{h})\rangle \\ &\text{where} \\ &add_{pd^{pos}}(\mathfrak{h}) = \{\langle l^c,t\rangle \mid \exists \langle ep,t\rangle \in \boldsymbol{\epsilon}(\mathfrak{h}) : l^c \in c(ep) \wedge \langle l^e,t+1\rangle \in \boldsymbol{\kappa}(\mathfrak{h}) \wedge \\ &\qquad\qquad \langle \overline{l^e},t\rangle \in \boldsymbol{\kappa}(\mathfrak{h}) \wedge (\forall \langle ep',t\rangle \in \boldsymbol{\epsilon}(\mathfrak{h}) : (ep' = ep \vee e(ep') \neq l^e))\} \\ &\text{with } l^e = e(ep) \end{aligned} \tag{12}$$

The function pd^{neg} (13) describes *negative postdiction*. This is the inference that knowledge about one yet unknown condition

literal of an effect proposition is gained if the effect is known not hold after an action a and all other condition literals are known to hold before the action. Example 11 in Appendix A.2 illustrates how knowledge is gained through postdiction.

$$
\begin{aligned}
&pd^{neg}(\mathfrak{h}) = \langle \boldsymbol{\alpha}(\mathfrak{h}), \boldsymbol{\kappa}(\mathfrak{h}) \cup add_{pd^{neg}}(\mathfrak{h}) \rangle \\
&\text{where} \\
&add_{pd^{neg}} = \{\langle \overline{l_u^c}, t \rangle \,|\exists \langle ep, t \rangle \in \boldsymbol{\epsilon}(\mathfrak{h}) : l_u^c \in c(ep) \wedge \langle \overline{l^e}, t+1 \rangle \in \boldsymbol{\kappa}(\mathfrak{h}) \wedge (\forall l^c \in c(ep) \setminus l_u^c : \langle l^c, t \rangle \in \boldsymbol{\kappa}(\mathfrak{h}))\} \\
&\text{with } l^e = e(ep)
\end{aligned}
\tag{13}
$$

As an example consider an action `open_door`, with the single effect proposition **causes** (`open_door`, `is_open`, $\neg$`jammed`). Assume that this action is executed in the initial state, i.e. at step $t = 0$, and no other action is executed concurrently. Assume further, that nothing is known about the initial state. A sensing action `sense_open` is exectued after `open_door`, at step $t = 1$, and produces two branches: one with $\langle$`is_open`$, 1\rangle$ and one with $\langle \neg$`is_open`$, 1\rangle$. This triggers positive postdiction in the first branch, i.e. , $add_{pd^{pos}}$ produces $\langle \neg$`jammed`$, 0\rangle$. Negative postdiction is triggered in the second branch, i.e. , $add_{pd^{neg}}$ produces $\langle$`jammed`$, 0\rangle$. For a more detailed illustration consider the step from $\tilde{\mathfrak{h}}_2^{3^+}$ to $\tilde{\mathfrak{h}}_2^{4^+}$ in Example 10 in Appendix A.2.

3.2.7. Re-evaluation of Knowledge-level Effects

To apply all five inference mechanisms together we define an $evalOnce$ function that successively applies each of the inference mechanisms.

$$evalOnce(\mathfrak{h}) = pd^{neg}(pd^{pos}(cause(back(fwd(\mathfrak{h})))))) \tag{14}$$

A problem is that inference mechanisms may trigger each other in any order, so it is often not sufficient to apply **IM.1 – IM.5** only once. To this end, re-evaluation is defined recursively (15) until convergence is reached.

$$eval(\mathfrak{h}) = \begin{cases} \mathfrak{h} & \text{if } evalOnce(\mathfrak{h}) = \mathfrak{h} \\ eval(evalOnce(\mathfrak{h})) & \text{otherwise} \end{cases} \tag{15}$$

For an illustration of this fixed point approach, consider Example 12 in Appendix A.2. $eval$ converges in linear time, and is therefore reasonable, because given the restrictions in Sections 3.1 and 3.2.10, there exists only a linear number of elements in the knowledge history, and no element is ever removed from the knowledge history. For details see Lemma 3 in Appendix B.1.

3.2.8. Concurrent Conditional Plans

So far we considered only single state transition steps. In order to model more complex cases that imply several transitions we define concurrent conditional plans (CCP). A CCP is a combination of sequences of concurrent actions and `if-then-else` constructs, as formalized in Definition 4.[8]

Definition 4 (Concurrent conditional plan).

- *An empty sequence of actions (denoted by* `[]`*) is a CCP*
- *If $a_1, \ldots, a_n$ are actions, then* `[`a_1`||`$\ldots$`||`a_n`]` *is a CCP.*
- *If p_1 and p_2 are concurrent conditional plans, then* `[`p_1`;`p_2`]` *is a CCP.*
- *If p_1 and p_2 are concurrent conditional plans and l is a fluent literal, then* `[if` l `then` p_1 `else` p_2`]` *is a CCP.*[9]

3.2.9. Extended Transition Function

We define an extended transition function $\widehat{\Psi}$ that maps a plan and a state to a set of states.

$$\widehat{\Psi}(p, \mathfrak{h}) = \begin{cases} \{\mathfrak{h}\} & \text{if } p = \texttt{[]} \\ \Psi(\{a_1 \ldots a_n\}, \mathfrak{h}) & \text{if } p = \texttt{[}a_1\texttt{||}\ldots\texttt{||}a_n\texttt{]} \\ \bigcup_{\mathfrak{h}' \in \widehat{\Psi}(p_1, \mathfrak{h})} \widehat{\Psi}(p_2, \mathfrak{h}') & \text{if } p = \texttt{[}p_1\texttt{;}p_2\texttt{]} \\ \widehat{\Psi}(p_1, \mathfrak{h}) & \text{if } p = \texttt{if } l \texttt{ then } p_1 \texttt{ else } p_2 \text{ and } \mathfrak{h} \models l \\ \widehat{\Psi}(p_2, \mathfrak{h}) & \text{if } p = \texttt{if } l \texttt{ then } p_1 \texttt{ else } p_2 \text{ and } \mathfrak{h} \not\models l \end{cases} \tag{16}$$

The extended transition function models *branching* as a reaction on respective sensing results. For illustration consider Example 7.

[8] Recall that we use a restricted form of concurrency where all actions have the same duration. Hence we consider only concurrent actions and not concurrent (sub-)plans.

[9] For notational convenience we allow to skip the `else`-part, i.e. `[if` l `then` p_1`]` is equivalent to `[if` l `then` p_1 `else []]`

Example 7. Extended transition function
Consider domain $\mathcal{D}_4$ and the CCP $p =$ `[open_door; sense_open; [if is_open then [drive]]]`.

$$\mathcal{D}_4 = \left\{ \begin{array}{r} \textbf{initially}\,(\neg\texttt{in\_room}) \\ \textbf{causes}\,(\texttt{drive}, \texttt{in\_room}, \{\texttt{is\_open}\}) \\ \textbf{causes}\,(\texttt{open\_door}, \texttt{is\_open}, \{\neg\texttt{jammed}\}) \\ \textbf{determines}\,(\texttt{sense\_open}, \texttt{is\_open}) \end{array} \right\} \tag{$\mathcal{D}_4$}$$

Given that the door is not jammed, the plan will lead to a state where the robot is in the destination room. This is achieved by the application of the extended transition function as follows: First `open_door` is executed, generating a single successor state. Then `sense_open` generates two h-states: one where the resulting h-state satisfies the condition `is_open` of the `drive` action, and another one where the condition is not satisfied. In the h-state where the condition is satisfied, the action `drive` is applied after sensing. In the other h-state, action execution ends after sensing. Hence, the extended transition function for p and $\mathcal{D}_4$ evaluates as follows:

$$\begin{aligned} \widehat{\Psi}(p_1, \mathfrak{h}_0) = \{ & \Psi(\texttt{drive}, \Psi(\texttt{sense\_open}, \Psi(\texttt{open\_door}, \mathfrak{h}_0))), \\ & \Psi(\texttt{sense\_open}, \Psi(\texttt{open\_door}, \mathfrak{h}_0))\} \end{aligned} \tag{17}$$

3.2.10. Consistency Issues

Our semantics is only reasonable under the following three consistency restrictions: (i) we forbid inconsistent initial states, (ii) we forbid that actions with contradictive effects are applied concurrently (e.g. opening and closing a door at the same time), and we forbid states that are not execute consistent, i.e. , if executability conditions of actions are always met (see Definition 3). To this end we describe consistency of h-states and consistency of CCP wrt. domains in the following Definitions 5 and 6.

Definition 5. *Consistent h-states*
An h-state $\mathfrak{h} = \langle \boldsymbol{\alpha}, \boldsymbol{\kappa} \rangle$ is consistent if $\forall \langle l, t \rangle \in \boldsymbol{\kappa} : \langle \bar{l}, t \rangle \notin \boldsymbol{\kappa}$, and if it is execute-consistent in the sense of Definifion 3.

Definition 6. *Consistency of plans wrt. domains*
A plan p is consistent wrt. a domain $\mathcal{D}$ if all h-states in the generated transition tree are consistent.

In the remainder of this paper we only consider consistent plans and domains.

4. Soundness and Complexity Results for Projection and Planning

$\mathcal{HPX}$ is designed to solve temporal projection and planning problems. In the following we define both types of problems, provide soundness results with a $\mathcal{PWS}$-based temporal epistemic action theory, and identify the computational complexity under reasonable restrictions.

4.1. The Projection Problem

The projection problem is that of predicting what an agent knows after executing a plan in a given domain. The problem is solved by applying the extended transition function and investigating all resulting leaf states of the transition tree.
Investigating the leafs can be done in two ways. One way is to ask what an agent *possibly knows* after executing a plan, and the other way is to ask what an agent *necessarily knows*. If a fact is known in all leaf states, the agent necessarily knows the fact; if a fact is known in at least one leaf state the agent possibly knows the fact. The latter depends on the outcome of sensing actions in the plan. This corresponds to the distinction between weak and strong planning (see e.g. [9]), and is captured in Definition 7. Note that the terminology to refer to strong and weak planning is diverse in literature. For example, Sardina et al. [45] call strong plans *adequate*.

Definition 7. *The projection problem*
Given a domain $\mathcal{D}$ and a plan p,

- *an agent* necessarily knows *that a pair $\langle l, t \rangle$ holds after executing p if (18) holds, i.e. if $\langle l, t \rangle$ is known in all leaf states of the transition tree.*

$$\forall \mathfrak{h} \in \widehat{\Psi}(p, \mathfrak{h}_0) : \mathfrak{h} \models \langle l, t \rangle \tag{18}$$

- *an agent* possibly knows *that a pair $\langle l, t \rangle$ holds after executing p if (19) holds, i.e. if $\langle l, t \rangle$ is known in at least one leaf state of the transition tree.*

$$\exists \mathfrak{h} \in \widehat{\Psi}(p, \mathfrak{h}_0) : \mathfrak{h} \models \langle l, t \rangle \tag{19}$$

In order to determine the computational complexity we first make some considerations pertaining to plans. Plans are solutions to real-world problems and therefore only feasible if their size is at most polynomial wrt. the problem specification. Towards this, the size of p is restricted to be polynomial wrt. the size of $\mathcal{D}$. In addition we restrict plans to have only a limited number of sensing actions.[10] The following Theorem 1 states, that, under such reasonable restrictions, solving the projection problem is polynomial.

Theorem 1 (Complexity of the $\mathcal{HPX}$ projection problem). *Given a domain $\mathcal{D}$ and a plan p of polynomial size, with a limited number of sensing actions; deciding whether (18) (resp. (19)) holds is polynomial.*

Proof:

- To solve the problem we call the extended transition function $\widehat{\Psi}(p, \mathfrak{h}_0)$ and iterate over all resulting leaf nodes to see whether a pair $\langle l, t\rangle$ is in the knowledge history of the leaf nodes. Since p is of polynomial size and since the number of sensing actions is limited, the number of nodes in the transition tree is of polynomial size wrt. $\mathcal{D}$.
- The transition function (6) is applied for each node (except for the leaf nodes), and is therefore called polynomially often.
- Calling the transition function (6) involves calling the re-evaluation function $eval$ (15), which in turn calls $evalOnce(\mathfrak{h})$ (14) until the h-state converged. Convergence is reached in polynomial time, because the size of the knowledge history $\boldsymbol{\kappa}(\mathfrak{h})$ is linear wrt. the depth of the plan p and we restrict p to be of polynomial size (for details see Lemma 1, Appendix B).
- A single re-evaluation step $evalOnce(\mathfrak{h})$ employs inference mechanisms **IM.1–IM.5**. These are all performed in polynomial time wrt. the size of the knowledge history (see Lemma 2, Appendix B for details). □

4.2. The Planning Problem

While for the projection problem a plan p is given and one asks for pairs $\langle l, t\rangle$, the planning problem is that of *deciding whether there exists a plan p*, such that, for a given domain $\mathcal{D}$ and a given pair $\langle l, t\rangle$, (18) (resp. (19)) holds. If (18) holds, $\langle l, t\rangle$ is known to hold in *all* leafs of the transition tree and p is a *strong plan*. If (19) holds, $\langle l, t\rangle$ is known to hold in *at least one* leaf of the transition tree and p is a *weak plan*. We refer the reader to [9] on more information on weak and strong planning, and to [39, 38] on certain problems concerning so-called *deadends* that occurr in strong planning problems.
The following theorem states that solving the planning problem is in NP, under the restriction that p is of polynomial size and the number of sensing actions is limited. Note that this is corresponds with the complexity of classical planning under equivalent restrictions, with complete knowledge and without sensing actions [1]. For a more general discussion on planning under partial observability, we refer to [44].

Theorem 2 (Complexity of the $\mathcal{HPX}$ planning problem). *Given a domain $\mathcal{D}$ and a pair $\langle l, t\rangle$ with $l \in \mathcal{L}_\mathcal{D}$ and finite $t \geq 0$; for p of polynomial size and with a limited number of sensing actions, deciding whether (20) (resp. 21) holds is in NP.*

$$\exists p : \Big(\forall \mathfrak{h} \in \widehat{\Psi}(p, \mathfrak{h}_0) : \mathfrak{h} \models \langle l, t\rangle\Big) \tag{20}$$

$$\exists p : \Big(\exists \mathfrak{h} \in \widehat{\Psi}(p, \mathfrak{h}_0) : \mathfrak{h} \models \langle l, t\rangle\Big) \tag{21}$$

Proof:
The theorem is a direct consequence of Theorem 1. Since p is restricted to be of polynomial size, and since deciding whether (18) (resp. (19)) holds is polynomial, one can solve (20) (resp. (21)) in NP time.

4.3. Soundness wrt. $\mathcal{PWS}$

In order to describe how $\mathcal{HPX}$ relates to traditional epistemic action theories which are based on a possible-worlds-semantics ($\mathcal{PWS}$) we investigate soundness wrt. the action language $\mathcal{A}_k$ by Son & Baral [48]. With respect to this, we have to consider that the $\mathcal{A}_k$ semantics is less expressive than $\mathcal{HPX}$ in that it does not support temporal knowledge. For example, propositions like "After sensing the location, the agent knows that the door was open before the driving." can not be made with $\mathcal{A}_k$. We address this issue in Sections 4.3.1 and 4.3.2, by defining an extended *Temporal Query Semantics* (called $\mathcal{A}_k^{TQS}$) that generalizes $\mathcal{A}_k$, and that allows for such propositions. Having defined the extended semantics $\mathcal{A}_k^{TQS}$, we can state the central soundness Theorem 3.

[10] Limiting the number of sensing actions means that the number of sensing actions is constant, and independent of the size of the plan. This is reasonable because we account for scenarios where actions are deterministic, so that the world does not have to be monitored continuously. Similar arguments can be found in [3].

4.3.1. An Action Language Based on Possible Worlds: The $\mathcal{A}_k$ Semantics

Let us first summarize the original $\mathcal{A}_k$ Semantics by Son & Baral [48]. It is defined via a transition function that maps actions and so-called *c-states* to c-states. A c-state δ is a tuple $\langle u, \Sigma \rangle$, where u is called a *state* and Σ is called a *k-state*. Informally, u represents a possible world and Σ represents the possible agent's belief wrt. u. A k-state Σ is a set of possible states s. A state (denoted u or s) is a set of fluents. If for a state s and a fluent f it holds that $f \in s$ then the value of f in s is $true$ and otherwise $false$. We require that c-states are *grounded*, i.e. that $u \in \Sigma$ for all c-states $\delta = \langle u, \Sigma \rangle$. Intuitively this means that the possible world u is among the worlds Σ the agent believes it could be in. For convenience we overload the $\models$-notation for a state s as follows:

$$\begin{aligned} (s \models f) &\Leftrightarrow (f \in s) \\ (s \models \neg f) &\Leftrightarrow (f \notin s) \\ (s \models \mathcal{L} &\Leftrightarrow (\forall l \in \mathcal{L} : s \models l) \end{aligned} \tag{22}$$

where $\mathcal{L}$ is a set of domain literals. Similarly we define for a k-state Σ:

$$(\Sigma \models l) \Leftrightarrow (\forall s \in \Sigma : s \models l) \tag{23}$$

Equation (23) reflects that a literal l is known to hold if it is $true$ in all possible worlds s in Σ. Given a domain description $\mathcal{D}$, one is interested in a set $\mathcal{M}$ of *valid models* of $\mathcal{D}$. A valid model $m = \langle \delta_0, \Phi \rangle$ is a pair of a *valid initial c-state* δ_0 and a transition function Ψ. An initial c-state is called *valid* if it does not contradict the initial knowledge defined in $\mathcal{D}$ (see [48] for details).
The transition function emerges from the effect propositions and knowledge propositions in $\mathcal{D}$. It maps an action a and a c-state to a c-state. It is defined with a case distinction as follows:

1. ***a* is a non-sensing action:** in this case the transition function is defined as:

$$\Phi(a, \langle u, \Sigma \rangle) = \langle Res(a, u), \{Res(a, s') | s' \in \Sigma\} \rangle \text{ where} \tag{24}$$

$$Res(a, s) = s \cup E_a^+(s) \setminus E_a^-(s) \text{ where} \tag{25}$$

$$E_a^+(s) = \{f \mid \exists ep \in \mathcal{EP}^a : e(ep) = f \wedge s \models c(ep)\}$$

$$E_a^-(s) = \{\neg f \mid \exists ep \in \mathcal{EP}^a : e(ep) = f \wedge s \models c(ep)\}$$

   Res (25) is a result function that reflects causation: if all conditions of an effect proposition ep hold (denoted $s \models c(ep)$), then the effect $e(ep)$ holds in the result.

2. ***a* is a sensing action:** in this case a has a knowledge proposition $\mathcal{KP}^a = f^s$ and the transition function is defined as:

$$\Phi(a, \langle u, \Sigma \rangle) = \langle u, \{s | (s \in \Sigma) \wedge (f^s \in s \Leftrightarrow f^s \in u)\} \rangle \tag{26}$$

   Intuitively, (26) rules out these possible worlds in Σ which do not coincide with the actual world u.

An illustration of the $\mathcal{A}_k$ semantics is provided in Appendix C, Example 13.

4.3.2. Temporal Query Semantics – $\mathcal{A}_k^{TQS}$

Our approach to make $\mathcal{A}_k$ capable of temporal reasoning is based on a re-evaluation step with the following intuition: let $\Sigma_0 = \{s_0^0, \ldots, s_0^{|\Sigma_0|}\}$ be the set of all possible initial states of a complete initial k-state wrt. a valid initial c-state δ_0 of an $\mathcal{A}_k$ domain $\mathcal{D}$. Whenever sensing happens, the transition function will remove some states from the k-state, i.e. $\Phi(a_n, \Phi(a_{n-1}, \cdots \Phi(a_1, \langle u_0, \Sigma_0 \rangle))) = \langle u_n, \Sigma_n \rangle$. To reason about the past, we *refine the set of possible initial states* and re-apply the result function to the refined set of initial states again. The refined set of initial states is the set of initial states which "survived" the transition of a sequence of actions.
For example, consider a sequence of n actions and say we are interested in the world state after the t-th action. Then we consider a *re-evaluated initial k-state*, denoted Σ_0^n, which consists of states $s_0 \in \Sigma_0$ such that for $s_n = Res(a_n, \cdots Res(a_1, s_0)$ it holds that $s_n \in \Sigma_n$. In other words, we consider these initial states s_0 of which the child states s_n "survived" the sensing actions.
Once we identify the re-evaluated initial k-state we apply the result function on each $s \in \Sigma_0^n$ up to the t-th action for this state. The resulting *re-evaluated k-state* is denoted Σ_n^t. If a fluent holds in all states $s_n^t \in \Sigma_n^t$, then after the n-th action, it is known that a fluent holds after the t-th action. This is formalized by Definitions 8 and 9. An illustration of the semantics is provided in Example 14, Appendix C.

Definition 8 (Re-evaluated initial k-state). *Let $\alpha = [a_1; \ldots; a_n]$ be a sequence of actions and $\delta_0 = \langle u_0, \Sigma_0 \rangle$ be a valid initial c-state such that $\langle u_n, \Sigma_n \rangle = \Phi(a_n, \Phi(a_{n-1}, \cdots \Phi(a_1, \delta_0)))$. We define a re-evaluated initial k-state, denoted Σ_n^0, as the set of*

initial belief states in Σ_0 which are valid after applying α.[11]

$$\Sigma_n^0 = \{s_0 | s_0 \in \Sigma_0 \wedge Res(a_n, Res(a_{n-1}, \cdots Res(a_1, s_0))) \in \Sigma_n\} \tag{27}$$

Re-evaluated c-states are defined in Definition 9: given a sequence of actions α, re-evaluated c-states are obtained by applying the $\mathcal{A}_k$ transition functions (24) and (26) on the re-evaluated initial k-state Σ_0^n.

Definition 9 (Re-evaluated c-states). *Let $\alpha = [a_1; \ldots; a_n]$ be a sequence of actions and $\delta_0 = \langle u_0, \Sigma_0 \rangle$ be a valid initial c-state, such that Σ_n^0 is a re-evaluated initial k-state according to Definition 8. We define a re-evaluated c-state, denoted δ_n^t as follows*

$$\delta_n^t = \langle u_n, \Sigma_n^t \rangle \tag{28}$$

$$\textit{where} \quad u_n = Res(a_n, u_{n-1}) \quad \textit{and} \quad \Sigma_n^t = \bigcup_{s \in \Sigma_n^{t-1}} Res(a_t, s) \tag{29}$$

$$\textit{with } 0 < t \leq n$$

Here, Σ_n^t is called the re-evaluated k-state.

4.3.3. Soundness of $\mathcal{HPX}$ wrt. $\mathcal{A}_k^{TQS}$

Now that we have defined $\mathcal{A}_k^{TQS}$– a semantics which is *(a)* based on the possible-worlds semantics and *(b)* can express temporal knowledge – we can formally relate $\mathcal{HPX}$ with a possible-worlds approach. Theorem 3 considers soundness of $\mathcal{HPX}$ wrt. ${\mathcal{A}_k}^{TQS}$ for the projection problem for a sequence of actions.

Soundness is defined wrt. an initial h-state $\mathfrak{h}_0$ and an arbitrary valid initial c-state of a domain. On the $\mathcal{A}_k^{TQS}$-side, the theorem considers one valid initial c-state u_0, i.e. one possible world which does not contradict the initial knowledge definitions. On the $\mathcal{HPX}$-side, we argue that there exists one h-state $\mathfrak{h}_n$ such that if a pair $\langle l, t \rangle$ is known to hold in $\mathfrak{h}_n$ then l is known to hold in the re-evaluated k-state Σ_n^t that results from the valid initial c-state u_0. A similar notion of soundness is presented for [48, Proposition 6, Lemma C.4].

Theorem 3 (Soundness of $\mathcal{HPX}$ wrt. ${\mathcal{A}_k}^{TQS}$). *Let $\alpha = [a_1; \ldots; a_n]$ be a sequence of actions and $\mathcal{D}$ a domain specification. Let $\langle u_0, \Sigma_0 \rangle$ be a valid grounded initial c-state of $\mathcal{D}$, such that with Definition 9 the re-evaluated c-state after $t \leq n$ actions is given as $\langle u_t, \Sigma_n^t \rangle = \Phi(a_t, \Phi(a_{t-1}, \cdots \Phi(a_1, \langle u_0, \Sigma_n^0 \rangle)))$. Then there exists at least one h-state $\mathfrak{h}_n \in \widehat{\Psi}(\alpha, \mathfrak{h}_0)$ such that for all literals l and all steps n, t with $0 \leq t \leq n$:*

$$(\mathfrak{h}_n \models \langle l, t \rangle) \Rightarrow (\Sigma_n^t \models l) \tag{30}$$

Proof sketch: [*Complete proof*: **Appendix C, Lemma 7**]
We prove the theorem by induction over the number of actions. The proof for the base step ($n = 0$) emerges from the definition of initial states. For the induction step, we consider the five individual inference mechanisms (9) – (13) and show that each IM is itself sound.

4.4. The Solvable Fragment of $\mathcal{HPX}$

The computational complexity of solving the plan-existence problem is in NP, and hence one level below the Σ_2^P complexity of the same problem in the $\mathcal{PWS}$-based $\mathcal{A}_k$ semantics of [48]. This comes at the price that $\mathcal{HPX}$ is not complete wrt. $\mathcal{PWS}$-based approaches. In [14, Sec. 4] we present a detailed empirical evaluation which shows that the solvable fragment of $\mathcal{HPX}$ is in most cases more than 90% of the full set of all projection problem instances (see [14, Fig. 1] for details). However, it is still interesting to look at a minimal example that demonstrates the incompleteness issue. To this end, consider the minimal Example 8, where knowledge is generated with a $\mathcal{PWS}$-based approach but not with $\mathcal{HPX}$. The example demonstrates that those problems that cannot be solved with $\mathcal{HPX}$, but that can be solved with $\mathcal{PWS}$-based approaches, are those where a fluent literal becomes known to hold because it obtains the same value in more than one possible world by a *non-sensing* action, and knowledge about the fluent literal is neither generated by inertia, nor by causation nor by postdiction.

[11] Consider that according to (25) $Res(a, s) = s$ if a is a sensing action.

Example 8. Incompleteness of $\mathcal{HPX}$
Consider the following action $\texttt{falsify}(f)$ that sets a fluent f to $\neg f$ if f is true.

$$\textbf{causes}\left(\texttt{falsify}(f), \neg f, \{f\}\right)$$

Assume that initially it is unknown whether f of $\neg f$ holds and compare how the execution of this action affects knowledge in 1. a $\mathcal{PWS}$-based approach and 2. in $\mathcal{HPX}$:

1. If f is initially unknown, then in a $\mathcal{PWS}$-based approach the agent's knowledge state is represented by a set of two possible worlds, denoted $\Sigma_0 = \{\{f\}, \{\neg f\}\}$. If action $\texttt{falsify}(f)$ is applied to Σ_0, then its effect proposition $\texttt{if } f \texttt{ then } \neg f$ is applied to both possible worlds resulting in a successor state Σ_1.

$$\Sigma_0 = \{\{f\}, \{\neg f\}\} \xrightarrow{\texttt{falsify}(f)} \Sigma_1 = \{\{\neg f\}, \{\neg f\}\}$$

   That is, the first possible world $\{f\}$ in state Σ_0 becomes $\{\neg f\}$ in the successor state Σ_1. Since a fluent literal is known to hold if it is true in all possible worlds, a $\mathcal{PWS}$-based semantics correctly represents that $\neg f$ is known to hold after executing $\texttt{falsify}(f)$.

2. With the h-approximation, the initial state would be $\mathfrak{h}_0 = \langle \emptyset, \emptyset \rangle$. Applying $\texttt{falsify}(f)$ evaluates as follows:

$$\Psi(\texttt{falsify}(f), \mathfrak{h}_0) = \{\langle \langle \texttt{falsify}(f), 0 \rangle, \emptyset \rangle\} = \mathfrak{h}_1 = \langle \boldsymbol{\alpha}_1, \boldsymbol{\kappa}_1 \rangle, \text{ with } \boldsymbol{\kappa}_1 = \emptyset$$

   That is, the agent does not acquire any new information about f after executing $\texttt{falsify}(f)$, i.e. $\boldsymbol{\kappa}_0 = \boldsymbol{\kappa}_1 = \emptyset$. The only way to generate knowledge in $\mathcal{HPX}$ is either through sensing or one of the inference mechanisms **IM.1** – **IM.5**, none of which apply in this case.

Fortunately, one can often work around such incompleteness issues in practice by re-formulating effect propositions appropriately. As an example consider the following non-conditional action $\texttt{falsify2}$, which might replace the original $\texttt{falsify}$:

$$\textbf{causes}\left(\texttt{falsify2}(f), \neg f, \emptyset\right)$$

In practice the outcome of $\texttt{falsify}(f)$ and $\texttt{falsify2}(f)$ is always identical, in that $\neg f$ will always hold after execution; $\mathcal{HPX}$ correctly generates knowledge if $\texttt{falsify2}(f)$ is used instead of $\texttt{falsify}(f)$. A general discussion about this approach is out of the scope of this paper.

5. Conclusion and Outlook

We present '*$\mathcal{HPX}$– h-approximation*', an epistemic action theory based on an approximation of the possible worlds semantics ($\mathcal{PWS}$) of knowledge. $\mathcal{HPX}$ provides native support for *postdiction* in an elaboration tolerant manner. It is also compuationally efficient, because within $\mathcal{HPX}$ the epistemic state of an agent can be represented with only a linear number of state variables. Other formalizations that support postdiction in an elaboration tolerant manner require an exponential number of variables. The lower number of state variables results in a lower computational complexity: the planning problem for $\mathcal{HPX}$ is in NP (see Theorem 2), while in traditional approaches it is Σ_2^P-complete [3]. The projection problem is tractable (see Theorem 1). Other approaches with a comparably low computational complexity require strong restrictions, as for example Bonet & Geffner [6], who do not allow situations where fluents that are initially unknown appear in the condition of an effect proposition. Similar strong restrictions can be found in [43].
Tractability of postdictive reasoning is achieved by explicitly representing the temporal dimension of knowledge. An added feature, basically a side-effect of the temporal dimension of knowledge, is that the $\mathcal{HPX}$ formalism also supports reasoning about the past, a feature which most other $\mathcal{PWS}$-based approaches of knowledge do not have. The only existing implemented system that also supports reasoning about the past is [31], but this has the disadvantage of exponentially many state variables (see Section 2). We show that the temporal dimension is important, because it allows $\mathcal{HPX}$ to solve problems which involve concurrent acting and sensing (see Example 9), and to model actions that sense and concurrently change a fluent's value (e.g. pulling the trigger of a gun causes one to know whether the gun was loaded, but at the same time the gun unloads.)
We show that the lower computational complexity of $\mathcal{HPX}$ comes at the cost that it is incomplete. However, in a follow-up paper [14], we demonstrate the usefulness of the solvable fragment and the overall theory.

We envision many possibilities to apply our theory in practice. Indeed, computing explanations and establishing hypotheses based on observations and partial knowledge is an important capability essential in a wide range of applications, such as the following:

- Robotic delibration, diagnosis and decision making require making sense of incomplete knowldge about the world. Here, additional postdictive information can be diagnosed by observing robot sensor output. We demonstrate this case in a case study in [14]. A video that illustrate the usefulness of postdition for abnormality detection in robotics can be found online.[12]
- Decision support systems in the field of medicine require the means to establish hypothesis based on background (domain) knowledge and narrative-based observations available from patient records.
- Dynamic geospatial systems is an emerging area of research where changes in qualitative spatio-temporal relationships between complex aggregate entities at the geospatial scale need to be analysed using high-level commonsense reasoning [5].
- Within spatial design cognition and computation (e.g., for architecture design), the need to perform diagnosis and compute counterfactual scenarios for incremental or iterative design refinement is clearly recognised.
- Assistive technologies in crime, forensics, and legal reasoning often involve perspective-dependent narratives of temporally grounded beliefs, and the formation of new beliefs or invalidation of old beliefs based on new knowledge.

In all the areas in the above, postdictive reasoning in the manner developed in this research can provide a computational basis to establish hypotheses with respect to domain-specific background theories. The postdictive reasoning capability in the manner supported by the approximate epistemic reasoner developed in this paper can 'plug' into a large-scale (hybrid) AI system. We emphasise, that in particular weak planning is important for real-world robotic domains, because of abnormalities: Weak planning can take into account that not all circumstances can be modelled, and is hence not restricted to closed artificial domains like string planning, which relies on the fact that the world model considers all possible abnormalities. A disadvantage of weak planning is that deadends can not be detected in advance, as e.g. done in the strong planning approach of Muise et al. [39, 38]. This could be solved by defining a metric for the likeliness of deadends in weak planning, which is another interesting future research direction.

Acknowledgements

Manfred Eppe acknowledges funding by the German Research Foundation (DFG) via the International Research Training Group on Semantic Integration of Geospatial Information (IRTG-SIGI), as well as funding by the Spanish National Research Council (CSIC) and the German Academic Exchange Service (DAAD). Mehul Bhatt acknowledges the funding and support of the German Research Foundation (DFG), www.dfg.de as part of the Spatial Cognition Research (SFB/TR 8). We thank Joohyung Lee for his review and feedback on this research; especially for his valuable comments concerning the soundness proofs. We thank Bernd Krieg-Brückner and other DFKI colleagues for providing support with the BAALL environment; in particular, Christian Mandel and Bernd Gersdorf provided support with the robotic infrastructure of the Rolland wheelchair robot. We also thank Brian Tietzen for his programming support and technical help in the context of the BAALL environment. This article has been built upon the following papers at LPNMR 2013 and COMMONSENSE 2013:

- "Approximate Epistemic Planning with Postdiction as Answer-Set Programming" [15]
- "Narrative based postditive reasoning for Cognitive Robotics" [13]

We thank the anonymous reviewers of these papers for their constructive comments and suggestions. We also thank the anonymous editors and reviewers of previous versions of this article.

[12] http://www.manfred.eppe.eu/the-hpx-online-planning-system/

References

[1] Bäckström, C., & Nebel, B. (1995). Complexity results for SAS+ planning. *Computational Intelligence*, *11*, 625—-655.

[2] Baral, C. (2003). *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

[3] Baral, C., Kreinovich, V., & Trejo, R. (2000). Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence*, *122*, 241–267.

[4] Bertoli, P., Cimatti, A., Pistore, M., Roveri, M., & Traverso, P. (2001). MBP : a Model Based Planner. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

[5] Bhatt, M., & Wallgruen, J. O. (2013). Geospatial Narratives and their Spatio-Temporal Dynamics: Commonsense Reasoning for High-level Analyses in Geographic Information Systems. In D. Rocchini (Ed.), *Draft of article to appear at: ISPRS International Journal of Geo-Information; Special Issue on: Geospatial Monitoring and Modelling of Environmental Change*. arXiv – Cornell University Library. (draft available on arXiv).

[6] Bonet, B., & Geffner, H. (2011). Planning under Partial Observability by Classical Replanning: Theory and Experiments. *IJCAI Proceedings*, .

[7] Brachman, R. J., & Levesque, H. J. (2004). *Knowledge Representation and Reasoning*. Elsevier.

[8] Carnap, R. (1956). *Meaning and Necessisity: A Study in Semantics and Modal Logic*. The University of Chicago Press.

[9] Cimatti, A., Pistore, M., Roveri, M., & Traverso, P. (2003). Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, .

[10] Davis, E. (1990). *Representations of Common Sense Knowledge*. The Morgan Kaufmann Series in Representation and Reasoning. Elsevier Science Limited.

[11] Demolombe, R., & del Pilar Pozos Parra, M. (2000). A simple and tractable extension of situation calculus to epistemic logic. In *International Symposium on Foundations of Intelligent Systems*.

[12] van Ditmarsch, H., van der Hoek, W., & Kooi, B. (2007). *Dynamic Epistemic Logic*. Springer.

[13] Eppe, M., & Bhatt, M. (2013). Narrative based Postdictive Reasoning for Cognitive Robotics. In *11th International Symposium on Logical Formalizations of Commonsense Reasoning*.

[14] Eppe, M., & Bhatt, M. (2015). Approximate postdictive reasoning with answer set programming. *Journal of Applied Logic*, *this issue*.

[15] Eppe, M., Bhatt, M., & Dylla, F. (2013). Approximate Epistemic Planning with Postdiction as Answer-Set Programming. In *Prodeedings of the 12th International Conference on Logic Programming and Nonmonotonic Reasoning*.

[16] Fagin, R., Halpern, J. Y., Moses, Y., & Vardi, M. Y. (1995). *Reasoning About Knowledge*. MIT Press.

[17] Fikes, R., & Nilsson, N. (1972). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, .

[18] de Giacomo, G., & Levesque, H. J. (1998). An Incremental Interpreter for High-Level Programs with Sensing. In *Working Notes of the 1998 AAAI Fall Symposium on Cognitive Robotics*.

[19] Hanks, S., & McDermott, D. (1987). Nonmonotonic logic and temporal projection. *Artificial Intelligence*, *33*, 379–412.

[20] van Harmelen, F., van Harmelen, F., Lifschitz, V., & Porter, B. (2007). *Handbook of Knowledge Representation*. San Diego, USA: Elsevier Science.

[21] Hintikka, J. (1962). *Logic and Belief: An Introduction to the Logic of the two Notions*. Ithaca: Cornell University Press.

[22] Hoffmann, J., & Brafman, R. I. (2005). Contingent planning via heuristic forward search with implicit belief states. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

[23] Hölldobler, S., & Schneeberger, J. (1990). A new deductive approach to planning. *New Generation Computing*, *8*, 225–244.

[24] Kahramanogullari, O., & Thielscher, M. (2003). A Formal Assessment Result for Fluent Calculus Using the Action Description Language Ak. In *German Conference on Artificial Intelligence*.

[25] Kowalski, R., & Sergot, M. (1986). A Logic-based calculus of events. *New generation computing*, *4*, 67–94.

[26] Kripke, S. (1963). Semantical Considerations on Modal Logic. *Acta Philisophica Fennica*, *16*, 83–94.

[27] Lakemeyer, G., & Levesque, H. J. (1998). AOL: a logic of acting, sensing, knowing, and only knowing. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*.

[28] Levesque, H. J. (1990). All I know: A study in autoepistemic Logic. *Artificial Intelligence*, *43*, 263–309.

[29] Liu, Y., & Levesque, H. J. (2005). Tractable reasoning with incomplete first-order knowledge in dynamic systems with context-dependent actions. In *International Joint Conference on Artificial Intelligence*.

[30] Lobo, J., Mendez, G., & Taylor, S. (2001). Knowledge and the Action Description Language A. *Theory and Practice of Logic Programming*, *1*, 129–184.

[31] Ma, J., Miller, R., Morgenstern, L., & Patkos, T. (2013). An Epistemic Event Calculus for ASP-based Reasoning About Knowledge of the Past, Present and Future. In *International Conference on Logic for Programming, Artificial Intelligence and Reasoning*.

[32] McCarthy, J. (1963). *Situations, Actions and Causal Laws*. Technical Report July Stanford Artificial Intelligence Project.

[33] McCarthy, J. (1998). Elaboration tolerance. In *International Symposium on Logical Formalizations of Commonsense Reasoning*.

[34] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., & Wilkins, D. (1998). *PDDL - The Planning Domain Definition Language*. Technical Report Yale Center for Computational Vision and Control.

[35] Miller, R., Morgenstern, L., & Patkos, T. (2013). Reasoning About Knowledge and Action in an Epistemic Event Calculus. In *International Symposium on Logical Formalizations of Commonsense Reasoning*.

[36] Moore, R. C. (1985). A formal theory of knowledge and action. In J. Hobbs, & R. C. Moore (Eds.), *Formal theories of the commonsense world*. Norwood, NJ: Ablex.

[37] Mueller, E. (2005). *Commonsense reasoning*. Morgan Kaufmann.

[38] Muise, C., Belle, V., & McIlraith, S. (2014). Computing Contingent Plans via Fully Observable Non-Deterministic Planning. In *AAAI Conference on Artificial Intelligence*.

[39] Muise, C., McIlraith, S. a., & Beck, J. C. (2012). Improved non-deterministic planning by exploiting state relevance. *22nd International Conference on Automated Planning and Scheduling (ICAPS)*, (pp. 172–180).

[40] Patkos, T., & Plexousakis, D. (2009). Reasoning with Knowledge , Action and Time in Dynamic and Uncertain Domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

[41] Patkos, T., & Plexousakis, D. (2012). Epistemic and Causal Commonsense Reasoning in Partially Observable Dynamic Domains - From Sensors to Concepts. In *Bridges between the Methodological and Practical Work of the Robotics and Cognitive Systems Communities*. Springer.

[42] Petrick, R. P., & Bacchus, F. (2004). Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

[43] Petrick, R. P. a., & Levesque, H. J. (2002). Knowledge Equivalence in Combined Action Theories. In *Principles of Knowledge Representation and Reasoning*.

[44] Rintanen, J. (2004). Complexity of planning with partial observability. In *International Conference on Automated Plannign and Scheduling (ICAPS)*.

[45] Sardina, S., de Giacomo, G., Lespérance, Y., & Levesque, H. (2006). On the Limits of Planning over Belief States under Strict Uncertainty. In *Principles of Knowledge Representation and Reasoning*.

[46] Sardina, S., de Giacomo, G., Lespérance, Y., & Levesque, H. J. (2004). On the semantics of deliberation in IndiGolog – from theory to implementation. *Annals of Mathematics and Artificial Intelligence*, *41*, 259–299.

[47] Scherl, R. B., & Levesque, H. J. (2003). Knowledge, action, and the frame problem. *Artificial Intelligence*, *144*, 1–39.

[48] Son, T. C., & Baral, C. (2001). Formalizing sensing actions - A transition function based approach. *Artificial Intelligence*, *125*, 19–91.

[49] Thielscher, M. (1998). Introduction To The Fluent Calculus. *Linköping Electronic Articles in Computer and Information Science*, *3*.

[50] Thielscher, M. (2000). Representing the knowledge of a robot. In *International Conference on Principles of Knowledge Representation and Reasoning*.

[51] Thielscher, M. (2001). The concurrent, continuous fluent calculus. *Studia Logica*, *67*, 315–331.

[52] Thielscher, M. (2005). FLUX : A Logic Programming Method for Reasoning Agents. *Theory and Practice of Logic Programming*, *5*.

[53] To, S. T. (2011). On the impact of belief state representation in planning under uncertainty. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

[54] To, S. T. (2012). A New Approach to Contingent Planning Using A Disjunctive Representation in AND / OR forward Search with Novel Pruning Techniques. *Journal of Artificial Intelligence Research*, .

[55] Tu, P. H., Son, T. C., & Baral, C. (2007). Reasoning and planning with sensing actions, incomplete information, and static causal laws using answer set programming. *Theory and Practice of Logic Programming*, *7*, 377–450.

[56] Vlaeminck, H., Vennekens, J., & Denecker, M. (2012). A general representation and approximate inference algorithm for sensing actions. In *Advances in Artificial Intelligence - Australasian Joint Conference*.

Appendix A. Examples

Appendix A.1. A Modified Yale Shooting Scenario With Concurrent Acting and Sensing

We present an extended version of the well-known Yale shooting problem [19]. It shows that the temporal dimension of knowledge is required to reason about actions which sense a fluent's value and concurrently change this value: pulling the trigger of a gun causes one to sense (by hearing the explosion) whether the gun was loaded. At the same time the gun unloads. If the gun was loaded then one can conclude that the target is dead.

Example 9. Detailed version of the extended Yale shooting problem
Consider the following domain specification:

$$\textbf{initially}(\texttt{alive})$$
$$\textbf{causes}(\texttt{shoot}, \neg\texttt{loaded}, \emptyset)$$
$$\textbf{causes}(\texttt{shoot}, \neg\texttt{alive}, \{\texttt{loaded}\})$$
$$\textbf{determines}(\texttt{shoot}, \texttt{loaded})$$

The turkey is initially alive, but it is unknown whether the gun is loaded. Shooting unloads the gun and causes the turkey to be dead if it is loaded. In addition, shooting causes to know whether the gun was loaded. The task is to infer whether the turkey is dead after the shooting, depending whether or not the firing of the gun was perceived. In $\mathcal{HPX}$ this is possible, because sensing yields the value of a fluent *before* the action takes place. That is, if the action is executed at $t = 0$ then in the resulting state $t = 1$, knowledge about the loaded-ness at $t = 0$ is produced, even if this differs from the `loaded`-ness in the resulting state $t = 1$. According to Definition 2 about initial knowledge we have

$$\mathfrak{h}_0 = \langle \{\}, \{\langle \texttt{alive}, 0\rangle\}\rangle$$

The $\mathcal{HPX}$ transition function (6) evaluates as:

$$\begin{aligned}\Psi(\texttt{shoot}, \mathfrak{h}_0) =& \{\mathfrak{h}_1^+, \mathfrak{h}_1^-\}, \text{ where}\\ \mathfrak{h}_1^+ =& \mathit{eval}(\langle\langle \texttt{shoot}, 0\rangle, \{\langle \texttt{alive}, 0\rangle\} \cup \langle \texttt{loaded}, 0\rangle\rangle\\ \mathfrak{h}_1^- =& \mathit{eval}(\langle\langle \texttt{shoot}, 0\rangle, \{\langle \texttt{alive}, 0\rangle\} \cup \{\langle \neg\texttt{loaded}, 0\rangle\}\rangle\end{aligned}$$

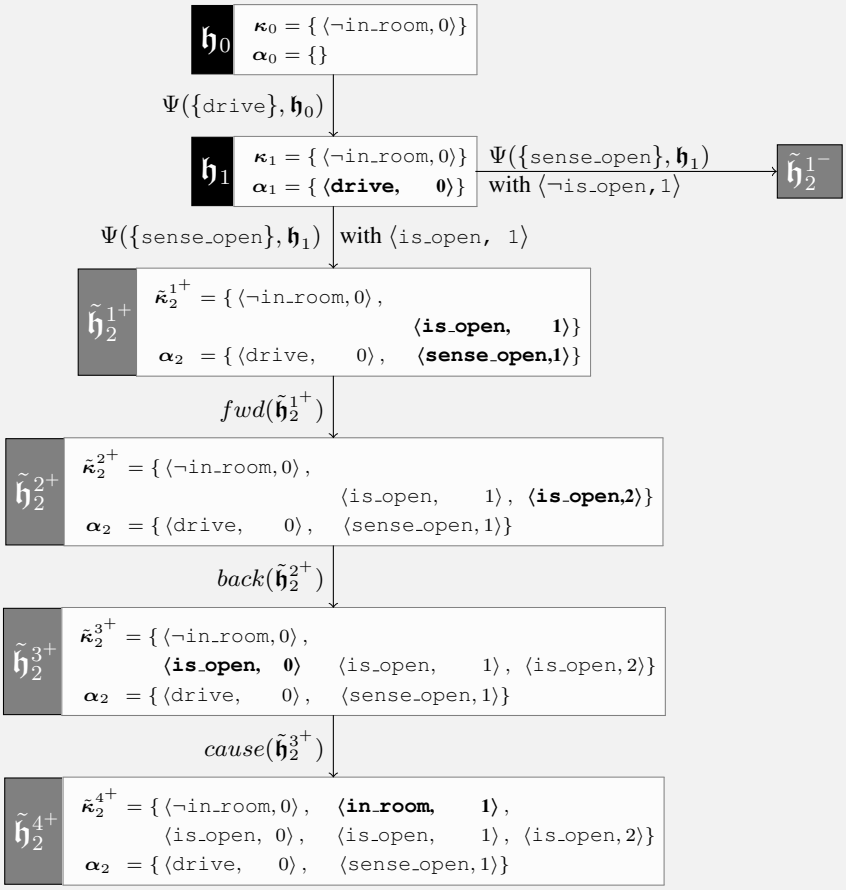
The eval function calls cause (11), and in the case of $\mathfrak{h}_1^+$ this correctly generates knowledge that the turkey is dead after shooting, i.e. $\mathfrak{h}_1^+ \models \langle \neg\texttt{alive}, 1\rangle$. In the case of $\mathfrak{h}_1^-$ cause correctly generates knowledge that the turkey is still `alive` after shooting: $\mathfrak{h}_1^- \models \langle \texttt{alive}, 1\rangle$.

Appendix A.2. Detailed Inference Examples

Example 10. Knowledge gain through sensing, causation and inertia
Consider domain $\mathcal{D}_2$ and the action sequence `[drive ; sense_open]`.

$$\mathcal{D}_2 = \left\{ \begin{array}{r} \textbf{initially}\,(\neg\texttt{in\_room}) \\ \textbf{causes}\,(\texttt{drive}, \texttt{in\_room}, \texttt{is\_open}) \\ \textbf{determines}\,(\texttt{sense\_open}, \texttt{is\_open}) \end{array} \right\} \tag{$\mathcal{D}_2$}$$

The state $\mathfrak{h}_0$ is the initial state where the door's open state is unknown, and where it is known that the robot is not in the destination room. The state transition from $\mathfrak{h}_0$ to $\mathfrak{h}_1$ represents that the robot drives through the door without actually knowing whether it is open.



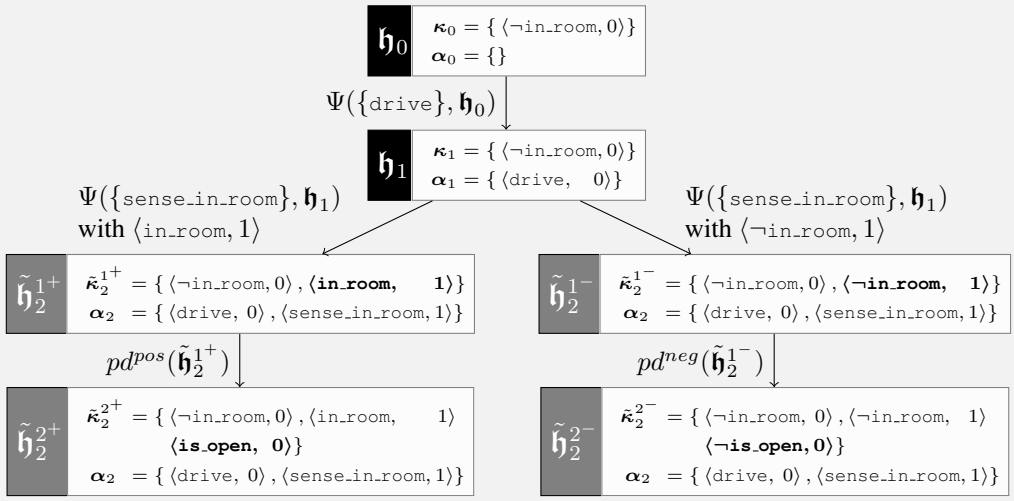
State $\mathfrak{h}_1$ is similar to $\mathfrak{h}_0$ because the agent does not know whether the door was open at the time of driving. Therefore it does not contain any *new* knowledge and we have that $\boldsymbol{\kappa}_0$ and $\boldsymbol{\kappa}_1$ are equal. `sense_open` generates two intermediate successor h-states: $\tilde{\mathfrak{h}}_2^{1+}$ contains the positive sensing outcome $\langle \texttt{is\_open}, 1\rangle$ and $\tilde{\mathfrak{h}}_2^{1-}$ contains the negative outcome $\langle \neg\texttt{is\_open}, 1\rangle$. For brevity we only consider the positive case. Forward inertia generates the next intermediate h-state $\tilde{\mathfrak{h}}_2^{2+}$. Consider that no action is applied that could change `is_open`. Therefore we have that $inertial(\texttt{is\_open}, 1, \tilde{\mathfrak{h}}_2^{2+})$ holds (8). Consequently, fwd (9) generates knowledge that the door is open in the future and adds $\langle \texttt{is\_open}, 2\rangle$ to $\tilde{\boldsymbol{\kappa}}_2^{2+}$. The case for next state $\tilde{\mathfrak{h}}_2^{3+}$ is similar: Since $\neg\texttt{is\_open}$ is inertial at step 0 we have that *back* (10) generates $\langle \texttt{is\_open}, 0\rangle$.
In the next step, causation generates knowledge. The action history $\boldsymbol{\alpha}_2$ contains information that `drive` was executed at step 0. Consequently there is an effect proposition in the effect history $\boldsymbol{\epsilon}(\boldsymbol{\alpha}_2)$ with an effect literal `in_room` and a condition literal `is_open` (see Definition 1). Therefore, since $\langle \texttt{open}, 0\rangle$ holds in $\tilde{\mathfrak{h}}_2^{3+}$, $cause(\tilde{\mathfrak{h}}_2^{3+})$ generates $\langle \texttt{in\_room}, 1\rangle$. Note that *cause* evaluates the effects of actions in a *retrospective manner*. That is, knowledge about the effect of driving is generated after sensing whether the door was open.[a]

[a] Note that this example considers that the robot first drives through a door without knowing its open-state and then senses the door's open-state. In practice it makes much more sense to first sense the door's open-state and then drive through the door. We consider this less sensible case any ways to illustrate how forward and backward inertia retrospectively generate knowledge.

Example 11. Knowledge gain through postdiction
Consider the domain $\mathcal{D}_3$ and the action sequence `[drive ; sense_in_room]`.

$$\mathcal{D}_3 = \left\{ \begin{array}{r} \textbf{initially}\,(\neg\texttt{in\_room}) \\ \textbf{causes}\,(\texttt{drive}, \texttt{in\_room}, \texttt{is\_open}) \\ \textbf{determines}\,(\texttt{sense\_in\_room}, \texttt{is\_room}) \end{array} \right\} \qquad (\mathcal{D}_3)$$

With the state transition from $\mathfrak{h}_0$ to $\mathfrak{h}_1$ no knowledge is gained because the condition of the `drive` action is not known to hold. For the next state transition we have two branches. That is, according to the transition function (6), `sense_in_room` generates two intermediate h-states denoted $\tilde{\mathfrak{h}}_2^{1^+}$ and $\tilde{\mathfrak{h}}_2^{1^-}$. In $\tilde{\mathfrak{h}}_2^{1^+}$ the robot is considered to be in the room, i.e. $\tilde{\mathfrak{h}}_2^{1^+} \models \langle \texttt{in\_room}, 1\rangle$ and in $\tilde{\mathfrak{h}}_2^{1^-}$ it is not in the room, i.e. $\tilde{\mathfrak{h}}_2^{1^-} \models \langle \neg\texttt{in\_room}, 1\rangle$.



In $\tilde{\mathfrak{h}}_2^{2^+}$ it is known that the robot was not in the room when the driving started ($\langle \neg\texttt{in\_room}, 0\rangle \in \tilde{\boldsymbol{\kappa}}_2^{1^+}$) but it was in the room after the driving ($\langle \texttt{in\_room}, 1\rangle \in \tilde{\boldsymbol{\kappa}}_2^{1^+}$). Consequently, positive postdiction generates knowledge that the condition of the `drive`-action was true, i.e. $\langle \texttt{is\_open}, 0\rangle \in add_{pd^{pos}}(\tilde{\mathfrak{h}}_2^{1^+})$.
In $\tilde{\mathfrak{h}}_2^{2^-}$ it is known that the robot was not in the room after the driving (i.e. $\langle \neg\texttt{in\_room}, 1\rangle \in \tilde{\boldsymbol{\kappa}}_2^{1^-}$). Consequently, negative postdiction generates knowledge that the condition of the `drive`-action was false, i.e. $\langle \neg\texttt{is\_open}, 0\rangle \in add_{pd^{neg}}(\tilde{\mathfrak{h}}_2^{1^-})$.

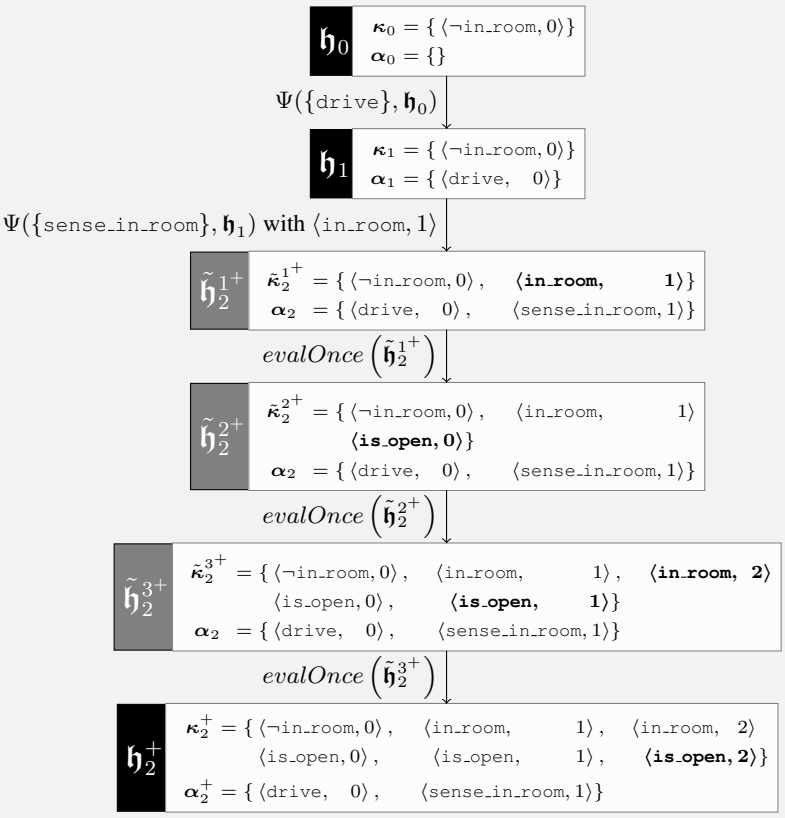
Example 12. Repeated Evaluation

Reconsider domain $\mathcal{D}_3$ and the sequence `[drive; sense_in_room]` as in Example 11.

$$\mathcal{D}_3 = \left\{ \begin{array}{r} \textbf{initially}\,(\neg\texttt{in\_room}) \\ \textbf{causes}\,(\texttt{drive}, \texttt{in\_room}, \texttt{is\_open}) \\ \textbf{determines}\,(\texttt{sense\_in\_room}, \texttt{is\_room}) \end{array} \right\} \tag{$\mathcal{D}_3$}$$

State $\mathfrak{h}_1$ which results from the `drive` action does not contain additional knowledge because the condition of the drive action (the door being open) is unknown. Sensing generates an intermediate successor state $\tilde{\mathfrak{h}}_2^{1+}$ state with $\tilde{\mathfrak{h}}_2^{1+} \models \langle \texttt{in\_room}, 1\rangle$. Thereafter, $evalOnce(\tilde{\mathfrak{h}}_2^{1+})$ (14) calls **IM.1** – **IM.5**. The only IM that produces knowledge in this first evaluation step is *positive postdiction* (12) – **IM.4** which adds a pair $\langle \texttt{is\_open}, 0\rangle$. This results in the next intermediate h-state $\tilde{\mathfrak{h}}_2^{2+}$.

In the next re-evaluation step, $evalOnce(\tilde{\mathfrak{h}}_2^{2+})$ calls **IM.1** – **IM.5** again, and *forward inertia* (9) – **IM.1** generates knowledge that the door is open during the sensing: $\langle \texttt{is\_open}, 1\rangle$. It also generates knowledge that the robot is in the room after the sensing: $\langle \texttt{in\_room}, 2\rangle$. A third application of $evalOnce$ results in the state $\mathfrak{h}_2^+$. Here, forward inertia generates knowledge that the door is open after the sensing: $\langle \texttt{is\_open}, 2\rangle$. The state $\mathfrak{h}_2^+$ is not an intermediate state because further application of $evalOnce$ will not produce any additional knowledge, i.e. the re-evaluation process converged.



Appendix B. Computational Properties of $\mathcal{HPX}$

This appendix contains Lemmata concerning the computational complexity properties of $\mathcal{HPX}$. These are required for the proofs of Theorems 1 and 2. As argued in Section 4, we assume in the following that the size of concurrent conditional plans (CCP) is of polynomial size wrt. $\mathcal{D}$ and that the number of sensing actions in the plan is limited.

Appendix B.1. Computational Complexity

Lemma 1 (Applying the transition function is polynomial). *Given a set of actions $\boldsymbol{A}$ and a consistent h-state $\mathfrak{h}$, applying the transition function $\Psi(\boldsymbol{A}, \mathfrak{h})$ (6) is polynomial.*

Proof: Recall the transition function (6):

$$\Psi(\mathbf{A}, \mathfrak{h}) = \bigcup_{k \in sense(\mathbf{A}^{ex}, \mathfrak{h})} eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(\mathfrak{h}) \cup k \rangle)$$

The transition function calls the $eval$ function (15) and conjoins its result with sensing results. Conjoining the sensing results is done in constant time and the number of potential sensing results is $|k| \leq 2$. Therefore the computational worst-time complexity is determined by $eval$.
To see that $eval$ is polynomial, consider the following: $eval(\mathfrak{h})$ calls $evalOnce(\mathfrak{h})$ until $\mathfrak{h}$ converged. We must therefore show that $evalOnce(\mathfrak{h})$ is (i) itself polynomial, and (ii) called at most polynomially often wrt. $\mathfrak{h}$. (i) is shown with Lemma 2 and (ii) is shown with Lemma 3. □

Lemma 2 (Applying $evalOnce$ (14) is polynomial). *Applying $evalOnce(\mathfrak{h})$ (14) is polynomial for an arbitrary h-state $\mathfrak{h}$ and a domain $\mathcal{D}$.*

Proof: $evalOnce$ (14) calls the five inference mechanisms (9) – (13), i.e. fwd, $back$, $causal$, pd^{pos} and pd^{neg}. All atomic operations (like concatenation of sets) in (9) – (13) are executed in linear or constant time. Quantifications in (9) – (13) are always either over the applied effect propositions, the condition literals in an effect proposition or over the elements in $\boldsymbol{\kappa}(\mathfrak{h})$ which are all sets of constant size wrt. $\mathfrak{h}$ and $\mathcal{D}$. Therefore $evalOnce(\mathfrak{h})$ is polynomial wrt. $\mathfrak{h}$ and $\mathcal{D}$. □

Lemma 3 (Function $evalOnce$ is called constantly often by $eval$). *Let $|\mathcal{L}_{\mathcal{D}}|$ be the number of literals in a domain $\mathcal{D}$ with the initial h-state $\mathfrak{h}_0$. Let p be a conditional plan and $\mathfrak{h} \in \widehat{\Psi}(p, \mathfrak{h}_0)$ be an arbitrary leaf state resulting from applying the extended transition function. Then $evalOnce(\mathfrak{h})$ (14) is called at most $|\mathcal{L}_{\mathcal{D}}| \cdot now(\mathfrak{h})$ times by $eval(\mathfrak{h})$ (15).*

Proof: It is easy to see that the maximum size of the knowledge history $\boldsymbol{\kappa}(\mathfrak{h})$ is $|\mathcal{L}_{\mathcal{D}}| \cdot now(\mathfrak{h})$. Lemma 4 proves monotonicity of $evalOnce$: that is, for a pair $\langle l, t \rangle$, if $\mathfrak{h} \models \langle l, t \rangle$ then $evalOnce(\mathfrak{h}) \models \langle l, t \rangle$. Therefore at most $|\mathcal{L}_{\mathcal{D}}| \cdot now(\mathfrak{h})$ changes can be made to $\boldsymbol{\kappa}(\mathfrak{h})$ until convergence is achieved. Thus, $evalOnce$ can only be called at most $|\mathcal{L}_{\mathcal{D}}| \cdot now(\mathfrak{h})$ times. □

Appendix B.2. Knowledge-persistence and Monotonicity of Re-evaluation

The following Lemmata capture that knowledge can not get lost, i.e. $\mathcal{HPX}$-agents do not "forget" knowledge.

Lemma 4 (Re-evaluation is monotonic). *Let $\mathcal{D}$ be a domain description with an initial state $\mathfrak{h}_0$ and p be a conditional concurrent plan. For all leaf states $\mathfrak{h} \in \widehat{\Psi(p, \mathfrak{h}_0)}$ it holds that*

$$\forall \langle l, t \rangle : (\mathfrak{h} \models \langle l, t \rangle \Rightarrow evalOnce(\mathfrak{h}) \models \langle l, t \rangle) \tag{B.1}$$

and

$$\forall \langle l, t \rangle : (\mathfrak{h} \models \langle l, t \rangle \Rightarrow eval(\mathfrak{h}) \models \langle l, t \rangle) \tag{B.2}$$

Proof: The proof of (B.1) follows from a simple syntactic investigation of five IM (9) – (13) which are called by $evalOnce$. Recall the definition of $evalOnce$ (14):

$$evalOnce(\mathfrak{h}) = pd^{neg}(pd^{pos}(cause(back(fwd(\mathfrak{h})))))$$

Let the following hold for an h-state $\mathfrak{h} = \langle \boldsymbol{\alpha}, \boldsymbol{\kappa} \rangle$:

$$\begin{aligned}
fwd(\mathfrak{h}) &= \tilde{\mathfrak{h}}_{fwd} = \langle \boldsymbol{\alpha}(\mathfrak{h}), \boldsymbol{\kappa}(\mathfrak{h}) \cup add_{fwd}(\mathfrak{h}) \rangle \\
back(fwd(\mathfrak{h})) &= \tilde{\mathfrak{h}}_{back} = \langle \boldsymbol{\alpha}(\mathfrak{h}), \boldsymbol{\kappa}(\mathfrak{h}_{fwd}) \cup add_{back}(\mathfrak{h}_{fwd}) \rangle \\
cause(back(fwd(\mathfrak{h}))) &= \tilde{\mathfrak{h}}_{cause} = \langle \boldsymbol{\alpha}(\mathfrak{h}), \boldsymbol{\kappa}(\mathfrak{h}_{back}) \cup add_{cause}(\mathfrak{h}_{back}) \rangle \\
pd^{pos}(cause(back(fwd(\mathfrak{h})))) &= \tilde{\mathfrak{h}}_{pd^{pos}} = \langle \boldsymbol{\alpha}(\mathfrak{h}), \boldsymbol{\kappa}(\mathfrak{h}_{cause}) \cup add_{pd^{pos}}(\mathfrak{h}_{cause}) \rangle \\
pd^{neg}(pd^{pos}(cause(back(fwd(\mathfrak{h}))))) &= \tilde{\mathfrak{h}}_{pd^{neg}} = \langle \boldsymbol{\alpha}(\mathfrak{h}), \boldsymbol{\kappa}(\mathfrak{h}_{pd^{pos}}) \cup add_{pd^{neg}}(\mathfrak{h}_{pd^{pos}}) \rangle
\end{aligned} \tag{B.3}$$

then $evalOnce(\mathfrak{h}) = \tilde{\mathfrak{h}}_{pd^{neg}}$.
From (B.3) we extract the following implications:

$$\begin{aligned}\forall \langle l,t\rangle : \big(\langle l,t\rangle \in \boldsymbol{\kappa}(\mathfrak{h}) \Rightarrow \langle l,t\rangle \in \boldsymbol{\kappa}(\mathfrak{h}_{fwd}) \Rightarrow \langle l,t\rangle \in \boldsymbol{\kappa}(\mathfrak{h}_{back}) \\ \Rightarrow \langle l,t\rangle \in \boldsymbol{\kappa}(\mathfrak{h}_{cause}) \Rightarrow \langle l,t\rangle \in \boldsymbol{\kappa}(\mathfrak{h}_{pd^{pos}}) \Rightarrow \langle l,t\rangle \in \boldsymbol{\kappa}(evalOnce(\mathfrak{h}))\big)\end{aligned} \tag{B.4}$$

It follows by (B.4) that

$$\forall \langle l,t\rangle : (\langle l,t\rangle \in \boldsymbol{\kappa}(\mathfrak{h}) \Rightarrow \langle l,t\rangle \in \boldsymbol{\kappa}(evalOnce(\mathfrak{h}))) \tag{B.5}$$

The definition of the $\models$ operator (5b) is :

$$\forall l,t : (\mathfrak{h} \models \langle l,t\rangle \Leftrightarrow \langle l,t\rangle \in \boldsymbol{\kappa}(\mathfrak{h})) \tag{B.6}$$

Consequently, $\forall \langle l,t\rangle : \mathfrak{h} \models \langle l,t\rangle \Rightarrow evalOnce(\mathfrak{h}) \models \langle l,t\rangle$ (B.1) is true.
The proof of (B.2) follows from (B.1) and the recursive definition of $eval$ (15). □

Lemma 5 (Knowledge-persistence for CCP). *Consider an h-state $\mathfrak{h}$ a concurrent conditional plan p. Then (B.7) holds:*

$$\forall l,t,\mathfrak{h}' \in \widehat{\Psi}(p,\mathfrak{h}) : \big(\mathfrak{h} \models \langle l,t\rangle \Rightarrow \mathfrak{h}' \models \langle l,t\rangle\big) \tag{B.7}$$

Proof: We perform induction over the structure of a CCP p.

1. $p = []$
   In this case $\widehat{\Psi}([],\mathfrak{h}) = \{\mathfrak{h}\}$ and the lemma trivially holds.
2. $p = [a_1||\ldots||a_n]$
   In this case $\widehat{\Psi}([a_1||\ldots||a_n],\mathfrak{h}) = \Psi(\{a_1,\ldots,a_n\},\mathfrak{h})$. Recall the transition function (6):

   $$\Psi(\{a_1,\ldots,a_n\},\mathfrak{h}) = \bigcup_{k \in sense(\{a_1,\ldots,a_n\},\mathfrak{h})} eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(\mathfrak{h}) \cup k\rangle) \tag{B.8}$$

   If follows trivially that (B.9) holds.

   $$\forall l,t,k \in sense(\{a_1,\ldots,a_n\},\mathfrak{h}) : \big(\mathfrak{h} \models \langle l,t\rangle \Rightarrow \boldsymbol{\kappa}(\mathfrak{h}) \cup k \models \langle l,t\rangle\big) \tag{B.9}$$

   Lemma 4 shows that $eval$ is monotonic, i.e. (B.10) holds.

   $$\forall l,t,k \in sense(\{a_1,\ldots,a_n\},\mathfrak{h}) : \big(\mathfrak{h} \models \langle l,t\rangle \Rightarrow eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(\mathfrak{h}) \cup k\rangle) \models \langle l,t\rangle\big) \tag{B.10}$$

   Consequently, (B.8) is true.
3. $p = [p_1; p_2]$ where p_1, p_2 are CCP
   This follows directly from the induction hypothesis and the extended transition function (16).
4. $p = [\texttt{if } l \texttt{ then } p_1 \texttt{ else } p_2]$ where p_1, p_2 are CCP
   This follows directly from the induction hypothesis and the extended transition function (16).

□

Appendix B.3. Knowledge Producing Mechanisms

The following Lemma states that if knowledge is produced then it is either produced by sensing or one of the five inference mechanisms:

Lemma 6 (Knowledge producing mechanisms for single state transitions). *Given a domain $\mathcal{D}$, an h-state $\mathfrak{h}$ and a set of actions $\boldsymbol{A}$. Then for all h-states $\mathfrak{h}' \in \Psi(\boldsymbol{A},\mathfrak{h})$ it holds that a pair $\langle l,t\rangle$ can only be contained in the knowledge history $\boldsymbol{\kappa}(\mathfrak{h}')$ if and only if*

it was produced by sensing or one of the inference mechanisms **IM.1–IM.5** *(9) – (13). This is formally expressed as follows:*

$$\begin{aligned}\forall \langle l,t\rangle : \langle l,t\rangle \in \boldsymbol{\kappa}(\mathfrak{h}') \Leftrightarrow \\ \Big(& \langle l,t\rangle \in \boldsymbol{\kappa}(\mathfrak{h}) \\ \vee & \langle l,t\rangle \in add_{fwd}(\mathfrak{h}') \\ \vee & \langle l,t\rangle \in add_{back}(\mathfrak{h}') \\ \vee & \langle l,t\rangle \in add_{cause}(\mathfrak{h}') \\ \vee & \langle l,t\rangle \in add_{pd^{pos}}(\mathfrak{h}') \\ \vee & \langle l,t\rangle \in add_{pd^{neg}}(\mathfrak{h}') \\ \vee & \{\langle l,t\rangle\} \in sense(\{\boldsymbol{A},\mathfrak{h}\})\Big)\end{aligned} \tag{B.11}$$

Proof:
This follows from the constitution of the transition function (6) and the re-evaluation functions. Recall (6):

$$\Psi(\boldsymbol{A},\mathfrak{h}) = \bigcup_{k\in sense(\boldsymbol{A},\mathfrak{h})} eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(\mathfrak{h}) \cup k\rangle)$$

where $eval$ is recursively defined as follows:

$$eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(\mathfrak{h}) \cup k\rangle) = \begin{cases} \mathfrak{h} & \text{if } evalOnce(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(\mathfrak{h}) \cup k\rangle) = \langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(\mathfrak{h}) \cup k\rangle \\ eval(evalOnce(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(\mathfrak{h}) \cup k\rangle)) & \text{otherwise} \end{cases}$$

Recall $evalOnce$ (refeq:evalOnce):

$$evalOnce(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(\mathfrak{h}) \cup k\rangle) = pd^{neg}(pd^{pos}(cause(back(fwd(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(\mathfrak{h}) \cup k\rangle)))))$$

The $\Rightarrow$ direction of (B.11) follows directly from syntactic investigation of Ψ, $eval$ and the constitution of the inference mechanism functions fwd, $back$, $cause$, etc.: each of the five inference mechanisms calls one add-function (add_{fwd}, add_{back}, add_{cause}, etc.) which generates additional pairs $\langle l,t\rangle$. For example $fwd(\mathfrak{h}) = \langle \boldsymbol{\alpha}(\mathfrak{h}), \boldsymbol{\kappa}(\mathfrak{h}) \cup add_{fwd}(\mathfrak{h})\rangle$.
By Lemma 4 it holds that no knowledge is removed from the knowledge history of an h-state. Hence if for some h-state $\mathfrak{h}$ it holds that $\langle l,t\rangle \in add_{\boldsymbol{IM}}(\mathfrak{h})$, then $\langle l,t\rangle \in eval(\mathfrak{h})$ (where $\boldsymbol{IM}$ is either add,$back$,$cause$, etc.). This proves the $\Rightarrow$ direction of (B.11).
The $\Leftarrow$ direction of (B.11) follows from the observation that there is no other operation within the transition function which generates a pair $\langle l,t\rangle$. □

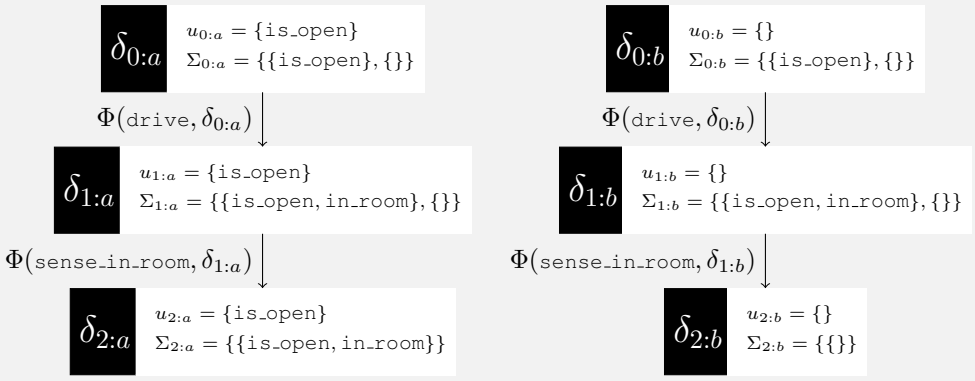
Appendix C. $\mathcal{HPX}$ and $\mathcal{A}_k^{TQS}$ Semantics

Appendix C.1. Illustrating Examples for $\mathcal{A}_k$ and $\mathcal{A}_k^{TQS}$

Example 13. **$\mathcal{PWS}$-based $\mathcal{A}_k$ semantics**
Consider the following domain description.

initially(¬in_room)
causes(drive, in_room, is_open)
determines(sense_in_room, in_room, ¬in_room)

A robot can execute an action drive to get into the living room if the door to the room is open. A fluent in_room denotes that it is in the living room and a fluent is_open denotes that the door is open. A sensing action sense_in_room can be executed to determine whether or not the robot is in the living room. The state transitions that we are interested in are depicted as follows:



Initially it is known that the robot is not in the living room and it is unknown whether the door is open. This results in two valid initial c-states: $\delta_{0:a} = \langle u_{0:a}, \Sigma_{0:a}\rangle$ and $\delta_{0:b} = \langle u_{0:b}, \Sigma_{0:b}\rangle$. Here, $u_{0:a}$ and $u_{0:b}$ represent two initial possible worlds where the door may be open or not, i.e. $u_{0:a} = \{\texttt{open}\}$ and $u_{0:b} = \{\}$. $\Sigma_{0:a}$ and $\Sigma_{0:b}$ represent two possible knowledge-states wrt. the possible worlds $u_{0:a}$ and $u_{0:b}$. Initially these are identical, i.e. $\Sigma_{0:a} = \Sigma_{0:b}$.
Applying the transition function (24) with drive results in the next states $\delta_{1:a}$ and $\delta_{1:b}$. Here we have that $u_{1:a} = \{\texttt{open}, \texttt{in\_room}\}$ and $u_{1:b} = \{\}$. In the second possible world $u_{1:b}$ the robot is stuck in front of a closed door. The knowledge-states in the two possible worlds are again identical because so far no sensing has happened, i.e. $\Sigma_{1:a} = \Sigma_{1:b}$.
To model the sensing action the transition function (26) generates the next c-states $\delta_{2:a}$ and $\delta_{2:b}$. The sensing "transfers" information about being in the living room or not from the actual world to the knowledge state. This is done by eliminating these states which are "incompatible" with the possible worlds $u_{1:a}$, resp. $u_{1:b}$. This causes two possible knowledge states wrt. each individual possible world, $\Sigma_{2:a} = \{\{\texttt{open}, \{\texttt{in\_room}\}\}$ and $\Sigma_{2:b} = \{\{\}\}$. In $\Sigma_{2:a}$ state it is known that the robot is in the living room and the door is open and in $\Sigma_{2:b}$ it is known that the robot is stuck at the closed door, not being in the living room.

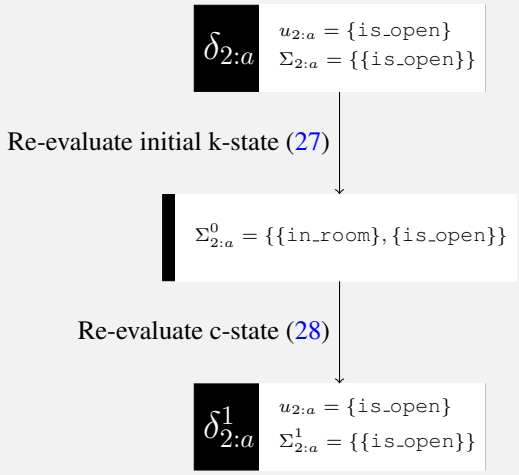
Example 14. ${\mathcal{A}_k}^{TQS}$ **semantics**
Consider the c-state $\delta_{2:a}$ from Example 13. The original $\mathcal{A}_k$ semantics allows one to infer that after the sensing the robot knows that the door is open and that it is in the living room, i.e. $\Sigma_{2:a} = \{\texttt{is\_open}, \texttt{in\_room}\}$. However, it does not allow one to infer whether the robot knows that the door was already open before the sensing, i.e. at a step $t = 1$. Our Temporal Query Semantics supports this inference as follows: Let $\delta_{2:a} = \langle u_{2:a}, \Sigma_{2:a}\rangle = \langle\{\texttt{is\_open}, \texttt{in\_room}\}, \{\{\texttt{is\_open}, \texttt{in\_room}\}\}\rangle$. Then applying (27) for re-evaluated initial k-states yields

$$\Sigma^0_{2:a} = \{s_0 \in \Sigma_{0:a} | Res(\texttt{sense\_in\_room}, Res(\texttt{drive}, s_0)) \in \Sigma_{2:a}\} = \{\texttt{is\_open}\}$$

The re-evaluation step (28) produces

$$\Sigma^1_{2:a} = Res(\texttt{drive}, \{\texttt{is\_open}\}) = \{\{\texttt{is\_open}, \texttt{in\_room}\}\}$$

In other words, $\mathcal{A}_k^{TQS}$ can express the temporal statement "after the sensing the robot knows that it was in the living room before the sensing".



Appendix C.2. Soundness proof
Recall Theorem 3 which states that $\mathcal{HPX}$ is sound wrt. the ${\mathcal{A}_k}^{TQS}$ semantics.

Theorem 3 (Soundness of $\mathcal{HPX}$ wrt. ${\mathcal{A}_k}^{TQS}$). *Let $\alpha = [a_1; \ldots; a_n]$ be a sequence of actions and $\mathcal{D}$ a domain specification. Let $\langle u_0, \Sigma_0\rangle$ be a valid grounded initial c-state of $\mathcal{D}$, such that with Definition 9 the re-evaluated c-state after $t \leq n$ actions is given as $\langle u_t, \Sigma^t_n\rangle = \Phi(a_t, \Phi(a_{t-1}, \cdots \Phi(a_1, \langle u_0, \Sigma^0_n\rangle)))$. Then there exists at least one h-state $\mathfrak{h}_n \in \widehat{\Psi}(\alpha, \mathfrak{h}_0)$ such that for all literals l and all steps n, t with $0 \leq t \leq n$:*

$$(\mathfrak{h}_n \models \langle l, t\rangle) \Rightarrow (\Sigma^t_n \models l) \tag{C.1}$$

Due to restrictions in $\mathcal{A}_k$, we forbid that actions can happen concurrently. That is, if an action has a knowledge proposition then it can not have an effect proposition. Recall the following notational conventions from Section 3:

- $\mathcal{D}$ is a domain specification.
- $\alpha_n = [a_1; \ldots; a_n]$ and $\alpha_{n+1} = [a_1; \ldots; a_n; a_{n+1}]$ are sequences of actions.
- $\mathfrak{h}_0$ is the initial h-state of $\mathcal{D}$.
- $\mathfrak{h}_n \in \widehat{\Psi}([a_1; \ldots; a_n], \mathfrak{h}_0)$ is an h-state which results from applying the extended transition function on $\mathfrak{h}_0$. Similar for $\mathfrak{h}_{n+1}$.
- $\delta_0 = \langle u_0, \Sigma_0\rangle$ is an arbitrary valid initial c-state of $\mathcal{D}$.
- $\delta_n = \langle u_n, \Sigma_n\rangle = \Phi(a_n, \Phi(a_{n-1}, \cdots \Phi(a_1, \delta_0)))$ is a c-state obtained by applying the $\mathcal{A}_k$ transition functions (24), (26). Similar for δ_{n+1}.

- Σ_n^0 and Σ_{n+1}^0 are the re-evaluated initial k-states as described in Definition 8:

$$\Sigma_n^0 = \{s_0 | s_0 \in \Sigma_0 \wedge Res(a_n, Res(a_{n-1}, \cdots Res(a_1, s_0))) \in \Sigma_n\}$$

  and similar for Σ_{n+1}^0.

- Σ_n^t and Σ_{n+1}^t are re-evaluated k-states as described in Definition 9:

$$\Sigma_n^t = \bigcup_{s \in \Sigma_n^0} Res(a_t, Res(a_{t-1}, \cdots Res(a_1, s)))$$

  with $0 \leq t \leq n$ and similar for Σ_{n+1}^t with $0 \leq t \leq n+1$.

With these conventions Theorem 3 is rewritten as Lemma 7.

Lemma 7 (Soundness of $\mathcal{HPX}$ wrt. $\mathcal{A}_k^{TQS}$ for sequences of actions).

$$\begin{aligned} &\forall n : \exists \mathfrak{h}_n \in \widehat{\Psi}(\alpha_n, \mathfrak{h}_0) : \forall l, t : \left(\mathfrak{h}_n \models \langle l, t \rangle \Rightarrow \Sigma_n^t \models l\right) \\ &\textit{with } t \leq n. \end{aligned} \tag{C.2}$$

Proof:
Induction over the number of actions n. The base step $n = 0$ is stated in Lemma 8. The induction step $(n \to n+1)$ is stated in Lemma 9. □

Appendix C.2.1. Base Step: Initial Knowledge
Lemma 8 considers soundness of knowledge in the initial state ($n = 0$). Since $t \leq n$ it holds that $t = 0$.

Lemma 8 (Soundness of the initial state). *Let $\mathcal{D}$ be a domain description and $\delta_0 = \langle u_0, \Sigma_0 \rangle$ a grounded valid initial c-state of $\mathcal{D}$ and h_0 be the initial h-state of D. Then (C.3) holds.*

$$\forall l : \mathfrak{h}_0 \models \langle l, 0 \rangle \Rightarrow \Sigma_0^0 \models l \tag{C.3}$$

Proof:
Definition 2 concerning the initial h-state h_0, Definition 8 concerning the re-evaluated initial k-state and the definition of initial knowledge in [48, p. 28, Definition 3] directly prove the Lemma. □

Appendix C.2.2. Induction Step: Knowledge Gain for Single State Transitions
The induction step reflects that after the $n+1$-th state transition is performed then there exists at least one h-state $\mathfrak{h}_{n+1}$ resulting from the state transition such that $\mathfrak{h}_{n+1} \models \langle l, t \rangle \Rightarrow \Sigma_{n+1}^t \models l$. This is formalized in Lemma 9.

Lemma 9 (Soundness of $\mathcal{HPX}$ wrt. $\mathcal{A}_k^{TQS}$ for single state transitions). *Let $\mathfrak{h}_n \in \widehat{\Psi}(\alpha_n, \mathfrak{h}_0)$ be an h-state such that (C.4) holds. Then (C.5) holds as well.*

$$\forall l, t : (\mathfrak{h}_n \models \langle l, t \rangle) \Rightarrow (\Sigma_n^t \models l) \tag{C.4}$$

$$\begin{aligned} &\exists \mathfrak{h}_{n+1} \in \Psi(a_{n+1}, \mathfrak{h}_n) : \\ &\forall l, t : (\mathfrak{h}_{n+1} \models \langle l, t \rangle) \Rightarrow (\Sigma_{n+1}^t \models l) \end{aligned} \tag{C.5}$$

with $t \leq n+1$.

Proof:
We fist make some substitutions and generalize over possible sensing results to eliminate the $\exists$-quantification over h-states. This allows us to perform induction over pairs $\langle l, t\rangle$.
(C.5)

Recall the $\mathcal{HPX}$-transition function (6).

$$\Psi(a_{n+1}, \mathfrak{h}_n) = \bigcup_{k \in sense(\{a_{n+1}\}, \mathfrak{h}_n)} eval(\langle \boldsymbol{\alpha}_{n+1}, \boldsymbol{\kappa}_n \cup k)\rangle)$$

where $\boldsymbol{\alpha}_{n+1} = \boldsymbol{\alpha}(\mathfrak{h}_n) \cup \{\langle a_{n+1}, now(\mathfrak{h}_n)\rangle\}$ and $\boldsymbol{\kappa}_n = \boldsymbol{\kappa}(\mathfrak{h}_n)$. Lemma 10 states that $now(\mathfrak{h}_n) = n$. With this we substitute in (C.5) and obtain (C.6)

$$\begin{aligned} &\exists \mathfrak{h}_{n+1} \in \bigcup_{k \in sense(\{a_{n+1}\}, \mathfrak{h}_n)} eval(\langle \boldsymbol{\alpha}_{n+1}, \boldsymbol{\kappa}_n \cup k\rangle) : \\ &\forall l, t : (\mathfrak{h}_{n+1} \models \langle l, t\rangle) \Rightarrow (\Sigma^t_{n+1} \models l) \end{aligned} \tag{C.6}$$

with $t \leq n+1$.

To prove (C.6) we make a generalization which eliminates the $\exists$-quantification over $\mathfrak{h}_{n+1}$. To this end consider the $sense$ function (7) which we rewrite as (C.7).

$$sense(\{a_{n+1}\}, \mathfrak{h}_n) = \begin{cases} \{\{\langle f^s, t^s\rangle\}, \{\langle \neg f^s, t^s\rangle\}\} & \text{if } \mathcal{KP}^{a_{n+1}} = f^s \wedge \\ & \quad \{\langle f^s, t^s\rangle, \langle \neg f^s, t^s\rangle\} \cap \boldsymbol{\kappa}_n = \emptyset \\ \{\emptyset\} & \text{otherwise} \end{cases} \tag{C.7}$$

where $t^s = now(\mathfrak{h}_n)$. By Lemma 10 it holds that $t^s = n$.
Let u_n denote the state which results from the application of the first n actions on the initial state u_0. This is formally expressed by (C.8).

$$u_n = Res(a_n, Res(a_{n-1}, \cdots Res(a_n, u_0))) \tag{C.8}$$

Since u_n is a set of fluent symbols there must be one sensing result $k \in sense(\{a_{n+1}\}, \mathfrak{h}_n)$ which corresponds to u_n. This correspondence is denoted by an auxiliary boolean function $corrSense(k, a_{n+1}, \mathfrak{h}_n, u_n)$ (C.9).

$$\begin{aligned} &corrSense(k, a_{n+1}, \mathfrak{h}_n, u_n) \Leftrightarrow \\ &\Big(k \in sense(\{a_{n+1}\}, \mathfrak{h}_n) \wedge \\ &\big((k = \{\emptyset\}) \vee (k = \{\langle f^s, n\rangle\} \wedge f^s \in u_n) \vee (k = \{\langle \neg f^s, n\rangle\} \wedge f^s \notin u_n)\big)\Big) \end{aligned} \tag{C.9}$$

where $f^s = \mathcal{KP}^{a_{n+1}}$. Consequently, in order to show that (C.6) holds it is sufficient to show that (C.10) holds.

$$\begin{aligned} \forall l, t, k : &\big((eval(\langle \boldsymbol{\alpha}_{n+1}, \boldsymbol{\kappa}_n \cup k\rangle) \models \langle l, t\rangle) \\ &\Rightarrow (\Sigma^t_{n+1} \models l)\big) \end{aligned} \tag{C.10}$$

with $t \leq n+1$ and $corrSense(k, a_{n+1}, \mathfrak{h}_n, u_n)$ holds.

To prove (C.10) we perform induction according to the structure of (B.11) in Lemma 6 over pairs $\langle l, t\rangle$.

Let $\mathfrak{h}^k_{n+1} = eval(\langle \boldsymbol{\alpha}_{n+1}, \boldsymbol{\kappa}_n \cup k\rangle$ such that $corrSense(k, a_{n+1}, \mathfrak{h}_n, u_n$ holds. Then (B.11) rewrites as follows:

$$\begin{aligned}\forall \langle l,t\rangle : \langle l,t\rangle \in \boldsymbol{\kappa}(\mathfrak{h}^k_{n+1}) \Leftrightarrow \Big(& \langle l,t\rangle \in \boldsymbol{\kappa}(\mathfrak{h}_n) \\ & \vee \{\langle l,t\rangle\} \in sense(\{a_{n+1}\}, \mathfrak{h}_n) \\ & \vee \langle l,t\rangle \in add_{fwd}(\mathfrak{h}^k_{n+1}) \\ & \vee \langle l,t\rangle \in add_{back}(\mathfrak{h}^k_{n+1}) \\ & \vee \langle l,t\rangle \in add_{cause}(\mathfrak{h}^k_{n+1}) \\ & \vee \langle l,t\rangle \in add_{pd^{pos}}(\mathfrak{h}^k_{n+1}) \\ & \vee \langle l,t\rangle \in add_{pd^{neg}}(\mathfrak{h}^k_{n+1})\Big)\end{aligned}$$

From (B.11) we extract the set of implications (C.11).

$$\forall \langle l,t\rangle : \langle l,t\rangle \in \boldsymbol{\kappa}^k_{n+1} \Leftarrow \langle l,t\rangle \in \boldsymbol{\kappa}(\mathfrak{h}_n) \tag{C.11a}$$
$$\forall \langle l,t\rangle : \langle l,t\rangle \in \boldsymbol{\kappa}^k_{n+1} \Leftarrow \{\langle l,t\rangle\} \in sense(\{a_{n+1}\}, \mathfrak{h}_n) \tag{C.11b}$$
$$\forall \langle l,t\rangle : \langle l,t\rangle \in \boldsymbol{\kappa}^k_{n+1} \Leftarrow \langle l,t\rangle \in add_{fwd}(\mathfrak{h}^k_{n+1}) \tag{C.11c}$$
$$\forall \langle l,t\rangle : \langle l,t\rangle \in \boldsymbol{\kappa}^k_{n+1} \Leftarrow \langle l,t\rangle \in add_{back}(\mathfrak{h}^k_{n+1}) \tag{C.11d}$$
$$\forall \langle l,t\rangle : \langle l,t\rangle \in \boldsymbol{\kappa}^k_{n+1} \Leftarrow \langle l,t\rangle \in add_{cause}(\mathfrak{h}^k_{n+1}) \tag{C.11e}$$
$$\forall \langle l,t\rangle : \langle l,t\rangle \in \boldsymbol{\kappa}^k_{n+1} \Leftarrow \langle l,t\rangle \in add_{pd^{pos}}(\mathfrak{h}^k_{n+1}) \tag{C.11f}$$
$$\forall \langle l,t\rangle : \langle l,t\rangle \in \boldsymbol{\kappa}^k_{n+1} \Leftarrow \langle l,t\rangle \in add_{pd^{neg}}(\mathfrak{h}^k_{n+1}) \tag{C.11g}$$

where $\boldsymbol{\kappa}^k_{n+1} = \boldsymbol{\kappa}(\mathfrak{h}^k_{n+1})$.
Two implications generate knowledge about a pair $\langle l,t\rangle$ independently from other pairs $\langle l',t'\rangle$ with $\langle l,t\rangle \neq \langle l',t'\rangle$. These are the cases (C.11a) and (C.11b). Since a soundness proof for these cases does not rely on the induction hypothesis we consider these cases for the base steps.
The remaining implications (C.11c), (C.11d), (C.11e), (C.11f) and (C.11g) produce $\langle l,t\rangle$ but rely on knowledge about $\langle l',t'\rangle$ with $\langle l,t\rangle \neq \langle l',t'\rangle$. For example a pair $\langle l,t\rangle$ is produced by the forward inertia function $add_{fwd}(\mathfrak{h}^k_{n+1})$ if $\langle l,t-1\rangle$ is known to hold in $\mathfrak{h}^k_{n+1}$. These implications are considered in the induction step because proving soundness relies on the induction hypothesis. The induction is complete because (B.11) is a bi-implication, i.e. all pairs $\langle l,t\rangle$ are reached.

Base Step 1 – (C.11a)

Recall (C.10):

$$\forall l,t,k : (\mathfrak{h}^k_{n+1} \models \langle l,t\rangle) \Rightarrow (\Sigma^t_{n+1} \models l)$$

with $t \leq n+1$.

Consider (C.11a): $\forall \langle l,t\rangle : \big(\mathfrak{h}^k_{n+1} \models \langle l,t\rangle \Leftarrow \langle l,t\rangle \in \boldsymbol{\kappa}(\mathfrak{h}_n)\big)$

$$\forall l,t : \big((\langle l,t\rangle \in \boldsymbol{\kappa}(\mathfrak{h}_n)) \Rightarrow (\Sigma^t_{n+1} \models l)\big) \tag{C.12}$$

with $t \leq n+1$.

By (C.4) the following holds: $\forall l,t : \big((\langle l,t\rangle \in \boldsymbol{\kappa}(\mathfrak{h}_n)) \Rightarrow (\Sigma^t_n \models l)\big)$. Therefore, to show that (C.12) holds it is sufficient to show that (C.13) holds.

$$\forall l,t : \big((\Sigma^t_n \models l) \Rightarrow (\Sigma^t_{n+1} \models l)\big) \text{with } t \leq n+1. \tag{C.13}$$

Lemma 11 states that (C.13) is true and we have proven base step 1.

Base Step 2 – (C.11b)

Recall (C.10):

$$\forall l, t, k : (corrSense(k, a_{n+1}, \mathfrak{h}_n, u_n) \wedge \mathfrak{h}^k_{n+1} \models \langle l, t\rangle) \Rightarrow (\Sigma^t_{n+1} \models l)$$

with $t \leq n+1$.

Consider (C.11b):
$\forall \langle l, t\rangle : \mathfrak{h}^k_{n+1} \models \langle l, t\rangle \Leftarrow \langle l, t\rangle \in \boldsymbol{\kappa}(\mathfrak{h}_n)$

$$\forall l, t, k : \big(corrSense(k, a_{n+1}, \mathfrak{h}_n, u_n) \wedge \{\langle l, t\rangle\} \in sense(\{a_{n+1}\}, \mathfrak{h}_n) \Rightarrow (\Sigma^t_{n+1} \models l)\big) \tag{C.14}$$

with $t \leq n+1$.

Reconsider $corrSense(k, a_{n+1}, \mathfrak{h}_n, u_n)$ (C.9):

$$\begin{aligned} &corrSense(k, a_{n+1}, \mathfrak{h}_n, u_n) \Leftrightarrow \\ &\Big(k \in sense(\{a_{n+1}\}, \mathfrak{h}_n) \wedge \\ &\big((k = \{\emptyset\}) \vee (k = \{\langle f^s, n\rangle\} \wedge f^s \in u_n) \vee (k = \{\langle \neg f^s, n\rangle\} \wedge f^s \notin u_n)\big)\Big) \end{aligned}$$

Reconsider the $sense$ function (C.7):

$$sense(\{a_{n+1}\}, \mathfrak{h}_n) = \begin{cases} \{\{\langle f^s, n\rangle\}, \{\langle \neg f^s, n\rangle\}\} & \text{if } \mathcal{KP}^a = f^s \wedge \\ & \quad \{\langle f^s, n\rangle, \langle \neg f^s, n\rangle\} \cap \boldsymbol{\kappa}_n = \emptyset \\ \{\emptyset\} & \text{otherwise} \end{cases}$$

Consequently, if $\{\langle l, t\rangle\} \in sense(\{a_{n+1}\}, \mathfrak{h}_n)$ then one of the following cases is true:

1. $\langle l, t\rangle = \langle f^s, n\rangle \wedge f^s \in u_n)$
2. $\langle l, t\rangle = \langle \neg f^s, n\rangle \wedge f^s \in u_n)$

   where $\mathcal{KP}^a = f^s$.
   We have to show that (C.14) holds in both cases. For brevity we show only case 1 (C.15). Case 2 is analogous.

$$\forall l, t : \big(\langle l, t\rangle = \langle f^s, n\rangle \wedge f^s \in u_n) \wedge \{\langle l, t\rangle\} \in sense(\{a_{n+1}\}, \mathfrak{h}_n) \Rightarrow (\Sigma^t_{n+1} \models l)\big) \tag{C.15}$$

with $t \leq n+1$ and $\mathcal{KP}^a = f^s$.

To show that (C.15) holds it is sufficient to show that (C.16) holds.

$$(f^s \in u_n) \Rightarrow (\Sigma^n_{n+1} \models f^s) \tag{C.16}$$

with $t \leq n+1$ and $\mathcal{KP}^a = f^s$.

By transition function for sensing actions (26):

$$(f^s \in u_n) \Rightarrow \big(\Sigma_{n+1} = \{s | (s \in \Sigma_n) \wedge (f^s \in s)\}\big)$$

It follows that (C.17) holds.

$$(f^s \in u_n) \Rightarrow (\forall s \in \Sigma_{n+1} : f^s \in s) \tag{C.17}$$

$$(\forall s \in \Sigma_{n+1} : f^s \in s) \Rightarrow (\Sigma^n_{n+1} \models f^s) \tag{C.18}$$

with $t \leq n+1$ and $\mathcal{KP}^a = f^s$.

By Lemma 12: $\Sigma_{n+1} = \Sigma^{n+1}_{n+1}$

$$(\forall s \in \Sigma^{n+1}_{n+1} : f^s \in s) \Rightarrow (\Sigma^n_{n+1} \models f^s) \tag{C.19}$$

with $t \leq n+1$ and $\mathcal{KP}^a = f^s$.

(C.19)

By (29): $\Sigma_{n+1}^{n+1} = \bigcup_{s \in \Sigma_{n+1}^{n}}^{R} es(a_{n+1}, s)$

Recall that we restrict that sensing actions do not have effect propositions. Since $\mathcal{KP}^{a_{n+1}} = f^s$ it holds that a_{n+1} is a sensing action and has no effect propositions. Therefore, by the $\mathcal{A}_k$ result function (25):

$$\forall s \in \Sigma_{n+1}^{n} : Res(a_{n+1}, s) = s$$

And consequently $\Sigma_{n+1}^{n} = \Sigma_{n+1}^{n+1}$.

$$(\forall s \in \Sigma_{n+1}^{n} : f^s \in s) \Rightarrow (\Sigma_{n+1}^{n} \models f^s) \tag{C.20}$$

with $t \leq n+1$ and $f^s = \mathcal{KP}^a$.

By definition of $\models$ (5): $\Sigma_{n+1}^{n} \models f^s \Leftrightarrow (\forall s \in \Sigma_{n+1}^{n} : f^s \in s)$.

This shows that (C.20) is true. Therefore base step 2 (C.12) is true.

Induction Step 1 – (C.11c)

Recall (C.10):

$$\forall l, t, k : (\mathfrak{h}_{n+1}^{k} \models \langle l, t \rangle) \Rightarrow (\Sigma_{n+1}^{t} \models l)$$

with $t \leq n+1$.

Consider (C.11c):

$$\forall \langle l, t \rangle : \mathfrak{h}_{n+1}^{k} \models \langle l, t \rangle \Leftarrow \langle l, t \rangle \in add_{fwd}(\mathfrak{h}_{n+1}^{k})$$

$$\forall l, t, k : \big(\langle l, t \rangle \in add_{fwd}(\mathfrak{h}_{n+1}^{k}) \Rightarrow (\Sigma_{n+1}^{t} \models l) \big) \tag{C.21}$$

with $t \leq n+1$.

Consider the definition of add_{fwd} (9):

$$add_{fwd}(\mathfrak{h}_{n+1}^{k}) = \{\langle l, t \rangle \,|(\langle l, t-1 \rangle \in \boldsymbol{\kappa}(\mathfrak{h}_{n+1}^{k}) \wedge inertial(l, t-1, \mathfrak{h}_{n+1}^{k}) \wedge t \leq now(\mathfrak{h}_{n+1}^{k}))\}$$

By Lemma 10 it holds that $now(\mathfrak{h}_{n+1}^{k}) = n+1$. To prove that (C.21) holds we prove that (C.22) holds.

$$\forall l, t, k : \Big(\big(\langle l, t-1 \rangle \in \boldsymbol{\kappa}_{n+1}^{k} \wedge inertial(l, t-1, \mathfrak{h}_{n+1}^{k}) \big) \Rightarrow (\Sigma_{n+1}^{t} \models l) \Big) \tag{C.22}$$

with $t \leq n+1$.

Consider the definition of $inertial$ (8):

$$inertial(l, t-1, \mathfrak{h}_{n+1}^{k}) \Leftrightarrow \forall \langle ep, t-1 \rangle \in \boldsymbol{\epsilon}(\mathfrak{h}_{n+1}^{k}) : \big(e(ep) = \bar{l}\big) \Rightarrow \Big(\exists l^c \in c(ep) : \langle \overline{l^c}, t-1 \rangle \in \boldsymbol{\kappa}(\mathfrak{h}_{n+1}^{k})\Big)$$

To prove that (C.22) holds we prove that (C.23) holds.

$$\begin{aligned} \forall l, t, k : \Big(\big(\langle l, t-1 \rangle \in \boldsymbol{\kappa}_{n+1}^{k} \wedge \\ \big(\forall ep : \big(\langle ep, t-1 \rangle \in \boldsymbol{\epsilon}(\mathfrak{h}_{n+1}^{k}) \Rightarrow \\ \big((e(ep) \neq \bar{l}) \vee (\exists l^c \in c(ep) : \langle \overline{l^c}, t-1 \rangle \in \boldsymbol{\kappa}_{n+1}^{k})\big)\big)\big) \\ \Rightarrow (\Sigma_{n+1}^{t} \models l) \Big) \end{aligned} \tag{C.23}$$

with $t \leq n+1$.

(C.23)

By induction hypothesis: $(\langle l, t-1\rangle \in \boldsymbol{\kappa}^k_{n+1}) \Rightarrow (\Sigma^{t-1}_{n+1} \models l)$.

By definition of $\models$ (23): $\Big(\Sigma^{t-1}_{n+1} \models l\Big) \Rightarrow \big(\forall s \in \Sigma^{t-1}_{n+1} : s \models l\big)$.

$$\forall l, t, k : \Big(\big((\forall s \in \Sigma^{t-1}_{n+1} : s \models l) \wedge \big(\forall ep : \big(\langle ep, t-1\rangle \in \boldsymbol{\epsilon}(\mathfrak{h}^k_{n+1}) \Rightarrow \big((e(ep) \neq \bar{l}) \vee (\exists l^c \in c(ep) : \langle \overline{l^c}, t-1\rangle \in \boldsymbol{\kappa}^k_{n+1}\big)\big)\big)\big) \Rightarrow (\Sigma^t_{n+1} \models l)\Big) \tag{C.24}$$

with $t \leq n+1$.

By the definition of effect histories (3) and the extended transition function (16): $\langle ep, t-1\rangle \in \boldsymbol{\epsilon}(\mathfrak{h}^k_{n+1}) \Rightarrow (ep \in \mathcal{EP}^{a_t})$. To show that (C.24) holds it is sufficient to show that (C.25) holds.

$$\forall l, t, k : \Big(\big((\forall s \in \Sigma^{t-1}_{n+1} : s \models l) \wedge \big(\forall ep \in \mathcal{EP}^{a_t} : \big((e(ep) \neq \bar{l}) \vee (\exists l^c \in c(ep) : \langle \overline{l^c}, t-1\rangle \in \boldsymbol{\kappa}^k_{n+1}\big)\big)\big) \Rightarrow (\Sigma^t_{n+1} \models l)\Big) \tag{C.25}$$

with $t \leq n+1$.

By definition of re-evaluated k-states (29): $\Sigma^t_{n+1} = \bigcup_{s \in \Sigma^{t-1}_{n+1}} Res(a_t, s)$.

By definition of $\models$ (23): $\Big(\bigcup_{s \in \Sigma^{t-1}_{n+1}} Res(a_t, s) \models l\Big) \Rightarrow (\forall s \in \Sigma^{t-1}_{n+1} : Res(a_t, s) \models l)$.

$$\forall l, t, k : \Big(\big((\forall s \in \Sigma^{t-1}_{n+1} : s \models l) \wedge \big(\forall ep \in \mathcal{EP}^{a_t} : \big((e(ep) \neq \bar{l}) \vee (\exists l^c \in c(ep) : \langle \overline{l^c}, t-1\rangle \in \boldsymbol{\kappa}^k_{n+1}\big)\big)\big) \Rightarrow (\forall s \in \Sigma^{t-1}_{n+1} : Res(a_t, s) \models l)\Big) \tag{C.26}$$

with $t \leq n+1$.

Case Distinction

To prove (C.26) we consider two cases for effect propositions ep, namely $e(ep) \neq \bar{l}$ and $\exists l^c \in c(ep) : \langle \overline{l^c}, t\rangle \in \boldsymbol{\kappa}^k_{n+1}$. We show that (C.26) holds in both cases.

1. $e(ep) \neq \bar{l}$ (Effect propositions do not have a complementary effect literal.)

   (C.26)

   We consider only cases where (C.27) holds.

   $$e(ep) \neq \bar{l} \tag{C.27}$$

   This simplifies (C.26) to (C.28).

   $$\forall l, t : \Big(\big(\forall s \in \Sigma^{t-1}_{n+1} : s \models l\big) \wedge \big(\forall ep \in \mathcal{EP}^{a_t} : e(ep) \neq \bar{l}\big) \Rightarrow (\forall s \in \Sigma^{t-1}_{n+1} : Res(a_t, s) \models l)\Big) \tag{C.28}$$

   with $t \leq n+1$.

(C.28)

Recall the $\mathcal{A}_k$ result function (25):

$$\begin{aligned} Res(a_t, s) &= s \cup E^+_{a_t}(s) \setminus E^-_{a_t}(s) \text{ where} \\ E^+_{a_t}(s) &= \{f | \exists ep \in \mathcal{EP}^{a_t} : e(ep) = f \wedge s \models c(ep)\} \\ E^-_{a_t}(s) &= \{f | \exists ep \in \mathcal{EP}^{a_t} : e(ep) = \neg f \wedge s \models c(ep)\} \end{aligned}$$

We distinguish two cases:

(a) $l = f \wedge \forall ep \in \mathcal{EP}^{a_t} : (e(ep) \neq \neg f)$
In this case $E^-_{a_t}(s) = \emptyset$. Therefore: $\forall s \in \Sigma^{t-1}_{n+1} : (s \models f \Rightarrow Res(a_t, s) \models f)$

(b) $l = \neg f \wedge \forall ep \in \mathcal{EP}^{a_t} : (e(ep) \neq f)$
In this case $E^+_{a_t}(s) = \emptyset$. Therefore: $\forall s \in \Sigma^{t-1}_{n+1} : (s \models \neg f \Rightarrow Res(a_t, s) \models \neg f)$

Consequently:

$$\forall l, t : \Big(\big(\forall ep \in \mathcal{EP}^{a_t} : (e(ep) \neq \bar{l}) \big) \Rightarrow \big(\forall s \in \Sigma^{t-1}_{n+1} : (s \models l \Rightarrow Res(a_t, s) \models l) \big) \Big) \tag{C.29}$$

It follows from (C.29) that (C.28) holds.

2. $(\exists l^c \in c(ep) : \langle \overline{l^c}, t-1 \rangle \in \boldsymbol{\kappa}^k_{n+1})$ (Effect propositions have a condition literal which is known not to hold.)

(C.26)

We consider only cases where (C.30) holds.

$$\exists l^c \in c(ep) : \langle \overline{l^c}, t-1 \rangle \in \boldsymbol{\kappa}^k_{n+1} \tag{C.30}$$

This simplifies (C.26) to (C.31).

$$\begin{aligned} \forall l, t, k : \Big(& \big(\forall s \in \Sigma^{t-1}_{n+1} : s \models l \big) \wedge \big(\forall ep \in \mathcal{EP}^{a_t} : (\exists l^c \in c(ep) : \langle \overline{l^c}, t-1 \rangle \in \boldsymbol{\kappa}^k_{n+1}) \big) \\ & \Rightarrow (\forall s \in \Sigma^{t-1}_{n+1} : Res(a_t, s) \models l) \Big) \end{aligned} \tag{C.31}$$

with $t \leq n+1$.

Recall the $\mathcal{A}_k$ result function (25):

$$\begin{aligned} Res(a_t, s) &= s \cup E^+_{a_t}(s) \setminus E^-_{a_t}(s) \text{ where} \\ E^+_{a_t}(s) &= \{f | \exists ep \in \mathcal{EP}^{a_t} : e(ep) = f \wedge s \models c(ep)\} \\ E^-_{a_t}(s) &= \{f | \exists ep \in \mathcal{EP}^{a_t} : e(ep) = \neg f \wedge s \models c(ep)\} \end{aligned}$$

We distinguish two cases:

(a) $l = f \wedge \forall ep \in \mathcal{EP}^{a_t} : (\exists l^c \in c(ep) : \langle \overline{l^c}, t-1 \rangle \in \boldsymbol{\kappa}^k_{n+1})$
By induction hypothesis we derive the following:

$$\forall l^c \in c(ep) : \big((\langle \overline{l^c}, t-1 \rangle \in \boldsymbol{\kappa}^k_{n+1}) \Rightarrow \Sigma^{t-1}_{n+1} \models \overline{l^c} \big)$$

In this case clearly $\forall s \in \Sigma^{t-1}_{n+1} : E^-_{a_t}(s) = \emptyset$.
Therefore: $\forall s \in \Sigma^{t-1}_{n+1} : (s \models f \Rightarrow Res(a_t, s) \models f)$.

(b) $l = \neg f \wedge \forall ep \in \mathcal{EP}^{a_t} : (\exists l^c \in c(ep) : \langle \overline{l^c}, t-1 \rangle \in \boldsymbol{\kappa}^k_{n+1})$
It follows similarly to case a) that $E^+_{a_t}(s) = \emptyset$.
Therefore: $\forall s \in \Sigma^{t-1}_{n+1} : (s \models \neg f \Rightarrow Res(a_t, s) \models \neg f)$

From a) and b) follows:

$$\forall l, t : \Big(\big(\forall ep \in \mathcal{EP}^{a_t} : (e(ep) \neq \bar{l}) \big) \Rightarrow \big(\forall s \in \Sigma^{t-1}_{n+1} : (s \models l \Rightarrow Res(a_t, s) \models l) \big) \Big) \tag{C.32}$$

It follows from (C.32) that (C.31) holds.

Induction Step 2 – (C.11d)
This is analogous to induction step 1 – (C.11c).

Induction Step 3 – (C.11e)
This is analogous to the following induction step 4 – (C.11f).

Induction Step 4 – (C.11f)
Recall (C.10):

$$\forall l, t, k : (\mathfrak{h}^k_{n+1} \models \langle l, t\rangle) \Rightarrow (\Sigma^t_{n+1} \models l)$$

with $t \leq n+1$.

Recall (C.11f):
$\forall \langle l, t\rangle : \mathfrak{h}^k_{n+1} \models \langle l, t\rangle \Leftarrow \langle l, t\rangle \in add_{pd^{pos}}(\mathfrak{h}^k_{n+1})$

$$\forall l, t, k : \big(\langle l, t\rangle \in add_{pd^{pos}}(\mathfrak{h}^k_{n+1}) \Rightarrow (\Sigma^t_{n+1} \models l)\big) \tag{C.33}$$

with $t \leq n+1$.

Consider the definition of $add_{pd^{pos}}$ (12):

$$add_{pd^{pos}}(\mathfrak{h}^k_{n+1}) = \{\langle l^c, t\rangle \,|\exists \langle ep, t\rangle \in \boldsymbol{\epsilon}(\mathfrak{h}^k_{n+1}) : \Big(l^c \in c(ep) \wedge \langle l^e, t+1\rangle \in \boldsymbol{\kappa}(\mathfrak{h}^k_{n+1}) \wedge \langle \overline{l^e}, t\rangle \in \boldsymbol{\kappa}(\mathfrak{h}^k_{n+1}) \\ \wedge \Big(\forall \langle ep', t\rangle \in \boldsymbol{\epsilon}(\mathfrak{h}^k_{n+1}) : \big(ep' = ep \vee e(ep') \neq l^e\big)\Big)\Big)$$

To prove that (C.33) holds we prove that (C.34) holds.

$$\forall l, t, k : \Big(\big(\exists ep : \big(\langle ep, t\rangle \in \boldsymbol{\epsilon}(\mathfrak{h}^k_{n+1}) \wedge e(ep) = l^e \wedge l \in c(ep) \wedge \langle l^e, t+1\rangle \in \boldsymbol{\kappa}(\mathfrak{h}^k_{n+1}) \wedge \langle \overline{l^e}, t\rangle \in \boldsymbol{\kappa}(\mathfrak{h}^k_{n+1}) \wedge \\ (\forall \langle ep', t\rangle \in \boldsymbol{\epsilon}(\mathfrak{h}^k_{n+1}) : (ep' = ep \vee e(ep') \neq l^e))\big)\big) \Rightarrow (\Sigma^t_{n+1} \models l)\Big) \tag{C.34}$$

with $t \leq n+1$.

By the definition of effect histories (3) and the extended transition function (16) it holds that:

$$\langle ep, t\rangle \in \boldsymbol{\epsilon}(\mathfrak{h}^k_{n+1}) \Rightarrow (ep \in \mathcal{EP}^{a_{t+1}})$$

To prove that (C.34) holds we prove that (C.35) holds.

$$\forall l, t, k : \Big(\big(\exists ep : \big(ep \in \mathcal{EP}^{a_{t+1}} \wedge e(ep) = l^e \wedge l \in c(ep) \wedge \langle l^e, t+1\rangle \in \boldsymbol{\kappa}(\mathfrak{h}^k_{n+1}) \wedge \langle \overline{l^e}, t\rangle \in \boldsymbol{\kappa}(\mathfrak{h}^k_{n+1}) \wedge \\ (\forall ep' \in \mathcal{EP}^{a_{t+1}} : (ep' = ep \vee e(ep') \neq l^e))\big)\big) \Rightarrow (\Sigma^t_{n+1} \models l)\Big) \tag{C.35}$$

with $t \leq n+1$.

By induction hypothesis:

$$(\langle l^e, t\rangle \in \boldsymbol{\kappa}^k_{n+1}) \Rightarrow (\Sigma^t_{n+1} \models l^e)$$

By definition of $\models$ (23):

$$\big(\Sigma^t_{n+1} \models l^e\big) \Rightarrow \big(\forall s \in \Sigma^t_{n+1} : s \models l^e\big)$$

$$\forall l, t : \Big(\big(\exists ep : \big(ep \in \mathcal{EP}^{a_{t+1}} \wedge e(ep) = l^e \wedge l \in c(ep) \wedge (\forall s \in \Sigma^{t+1}_{n+1} : s \models l^e) \wedge (\forall s \in \Sigma^t_{n+1} : s \models \overline{l^e}) \wedge \\ (\forall ep' \in \mathcal{EP}^{a_{t+1}} : (ep' = ep \vee e(ep') \neq l^e))\big)\big) \Rightarrow (\Sigma^t_{n+1} \models l)\Big) \tag{C.36}$$

with $t \leq n+1$.

(C.36)

Recall the definition of re-evaluated k-states (29): $\Sigma_{n+1}^{t+1} = \bigcup_{s \in \Sigma_{n+1}^{t}} Res(a_{t+1}, s)$. Therefore, to show that (C.36) holds it is sufficient to show that (C.37) holds.

$$\forall l, t : \Big(\big(\exists ep : \big(ep \in \mathcal{EP}^{a_{t+1}} \wedge e(ep) = l^e \wedge l \in c(ep) \wedge (\forall s \in \Sigma_{n+1}^{t} : Res(a_{t+1}, s) \models \overline{l^e}) \wedge (\forall s \in \Sigma_{n+1}^{t} : s \models l^e) \wedge \\ (\forall ep' \in \mathcal{EP}^{a_{t+1}} : (ep' = ep \vee e(ep') \neq l^e)) \big) \big) \Rightarrow (\Sigma_{n+1}^{t} \models l) \Big) \tag{C.37}$$

with $t \leq n+1$.

Simplification.

$$\forall l, t : \Big(\big(\exists ep : \big(ep \in \mathcal{EP}^{a_{t+1}} \wedge e(ep) = l^e \wedge l \in c(ep) \wedge (\forall s \in \Sigma_{n+1}^{t} : (Res(a_{t+1}, s) \models l^e \wedge s \models \overline{l^e})) \wedge \\ \big(\forall ep' \in \mathcal{EP}^{a_{t+1}} : (ep' = ep \vee e(ep') \neq l^e) \big) \big) \big) \Rightarrow (\Sigma_{n+1}^{t} \models l) \Big) \tag{C.38}$$

with $t \leq n+1$.

We rephrase towards a more compact representation:

$$\big(\exists ep : \big(ep \in \mathcal{EP}^{a_{t+1}} \wedge e(ep) = l^e \wedge (\forall ep' \in \mathcal{EP}^{a_{t+1}} : (ep' = ep \vee e(ep') \neq l^e)) \big) \big) \\ \Leftrightarrow \big(\exists! ep : \big(ep \in \mathcal{EP}^{a_{t+1}} \wedge e(ep) = l^e \big) \big)$$

$$\forall l, t : \Big(\big(\exists! ep : \big(ep \in \mathcal{EP}^{a_{t+1}} \wedge e(ep) = l^e \wedge l \in c(ep) \wedge (\forall s \in \Sigma_{n+1}^{t} : (Res(a_{t+1}, s) \models l^e \wedge s \models \overline{l^e})) \big) \big) \Rightarrow (\Sigma_{n+1}^{t} \models l) \Big) \tag{C.39}$$

with $t \leq n+1$.

Recall the $\models$ operator for k-states (23): $(\Sigma_{n+1}^{t} \models l) \Rightarrow (\forall s \in \Sigma_{n+1}^{t} : s \models l)$

$$\forall l, t : \Big(\big(\exists! ep : \big(ep \in \mathcal{EP}^{a_{t+1}} \wedge e(ep) = l^e \wedge l \in c(ep) \wedge (\forall s \in \Sigma_{n+1}^{t} : (Res(a_{t+1}, s) \models l^e \wedge s \models \overline{l^e})) \big) \big) \Rightarrow (\forall s \in \Sigma_{n+1}^{t} : s \models l) \Big) \tag{C.40}$$

with $t \leq n+1$.

For brevity we consider only the case where $e(ep) = l^e = f^e$. The case for $l^e = \neg f^e$ is similar. Recall (22): $s \models f^e \Leftrightarrow f^e \in s$.

$$\forall l, t : \Big(\big(\exists! ep : \big(ep \in \mathcal{EP}^{a_{t+1}} \wedge e(ep) = f^e \wedge l \in c(ep) \wedge (\forall s \in \Sigma_{n+1}^{t} : (f^e \in Res(a_{t+1}, s) \wedge f^e \notin s)) \big) \big) \Rightarrow (\forall s \in \Sigma_{n+1}^{t} : s \models l) \Big) \tag{C.41}$$

with $t \leq n+1$.

Recall the $\mathcal{A}_k$ result function (25):

$$Res(a_{t+1}, s) = s \cup E^{+}_{a_{t+1}}(s) \setminus E^{-}_{a_{t+1}}(s) \text{ where} \\ E^{+}_{a_{t+1}}(s) = \{ f | \exists ep \in \mathcal{EP}^{a_{t+1}} : e(ep) = f \wedge s \models c(ep) \} \\ E^{-}_{a_{t+1}}(s) = \{ f | \exists ep \in \mathcal{EP}^{a_{t+1}} : e(ep) = \neg f \wedge s \models c(ep) \}$$

Since we only consider cases with a positive effect literal $e(ep) = f^e$ the lower term $E^{-}_{a_{t+1}}(s)$ can be neglected. Consequently:

$$Res(a_{t+1}, s) = s \cup \{ f | \exists ep \in \mathcal{EP}^{a_{t+1}} : e(ep) = f \wedge s \models c(ep) \} \tag{C.42}$$

We substitute (C.42) in (C.41) and obtain (C.43).

$$\forall l, t : \Big(\big(\exists! ep : \big(ep \in \mathcal{EP}^{a_{t+1}} \wedge e(ep) = f^e \wedge l \in c(ep) \wedge \\ \big(\forall s \in \Sigma_{n+1}^{t} : (f^e \in (s \cup \{ f | \exists ep'' \in \mathcal{EP}^{a_{t+1}} : (e(ep'') = f \wedge s \models c(ep'')) \}) \wedge f^e \notin s) \big) \big) \\ \Rightarrow (\forall s \in \Sigma_{n+1}^{t} : s \models l) \Big) \tag{C.43}$$

with $t \leq n+1$.

We simplify:

$$(f^e \in (s \cup \{f | \exists ep'' \in \mathcal{EP}^{a_{t+1}} : (e(ep'') = f \wedge s \models c(ep''))\}) \wedge f^e \notin s)$$
$$\Leftrightarrow (\exists ep'' \in \mathcal{EP}^{a_{t+1}} : (e(ep'') = f^e \wedge s \models c(ep'')))$$

$$\forall l, t : \Big(\big(\exists ep! : \big(ep \in \mathcal{EP}^{a_{t+1}} \wedge e(ep) = f^e \wedge l \in c(ep) \wedge (\forall s \in \Sigma^t_{n+1} : (\exists ep'' \in \mathcal{EP}^{a_{t+1}} : (e(ep'') = f^e \wedge s \models c(ep'')))))\big) \Rightarrow (\forall s \in \Sigma^t_{n+1} : s \models l)\Big) \tag{C.44}$$

with $t \leq n+1$.

By definition of $\models$ (22):

$$(s \models c(ep'') \wedge l \in c(ep'')) \Rightarrow s \models l$$

$$\forall l, t : \Big(\big(\exists! ep : \big(ep \in \mathcal{EP}^{a_{t+1}} \wedge e(ep) = f^e \wedge l \in c(ep) \wedge (\forall s \in \Sigma^t_{n+1} : (\exists ep'' \in \mathcal{EP}^{a_{t+1}} : (e(ep'') = f^e \wedge s \models l)))\big)\big) \Rightarrow (\forall s \in \Sigma^t_{n+1} : s \models l)\Big) \tag{C.45}$$

with $t \leq n+1$.

We simplify.

$$\big(\exists! ep : \big(ep \in \mathcal{EP}^{a_{t+1}} \wedge e(ep) = f^e \wedge l \in c(ep) \wedge (\forall s \in \Sigma^t_{n+1} : (\exists ep'' \in \mathcal{EP}^{a_{t+1}} : (e(ep'') = f^e \wedge s \models l)))\big)\big)$$
$$\Leftrightarrow$$
$$\big(\exists! ep : \big(ep \in \mathcal{EP}^{a_{t+1}} \wedge e(ep) = f^e \wedge l \in c(ep) \wedge (\forall s \in \Sigma^t_{n+1} : s \models l)\big)\big)$$

$$\forall l, t : \Big(\big(\exists! ep : \big(ep \in \mathcal{EP}^{a_{t+1}} \wedge e(ep) = f^e \wedge l \in c(ep) \wedge (\forall s \in \Sigma^t_{n+1} : s \models l)\big)\big) \Rightarrow (\forall s \in \Sigma^t_{n+1} : s \models l)\Big) \tag{C.46}$$

with $t \leq n+1$.

It is easy to see that (C.46) is true.

Induction Step 5 – (C.11g)
This is analogous to induction step 4 – (C.11f). □

Appendix C.3. Additional Lemmata
Appendix C.3.1. Number of Steps
The following Lemma 10 concerns the number of state transitions for an h-state $\mathfrak{h}$.

Lemma 10 (Step number for sequences of actions). *Given a domain $\mathcal{D}$ with an initial h-state $\mathfrak{h}_0$ and a sequence of actions $\alpha_n = [a_1 || \cdots || a_n]$. Then the following holds:*

$$\forall \mathfrak{h} \in \widehat{\Psi}(\alpha_n, \mathfrak{h}_0) : now(\mathfrak{h}) = n \tag{C.47}$$

Proof:
The Lemma follows directly from the extended $\mathcal{HPX}$-transition function (16) and the $\mathcal{HPX}$-transition function (6).

Appendix C.3.2. Knowledge Persistence
The following Lemmata state that in the temporal query semantics ${\mathcal{A}_k}^{TQS}$, knowledge itself is persistent (Lemma 11) an that knowledge about the presence is no affected by re-evaluation (Lemma 12).

Lemma 11 (Knowledge persistence in re-evaluated c-states). *Given a domain $\mathcal{D}$, a valid initial c-state $\delta_0 = \langle u_0, \Sigma_0 \rangle$ a sequence of actions $\alpha = [a_1, \ldots, a_n, a_{n+1}]$ which produces re-evaluated k-states Σ_n^t and Σ_{n+1}^t according to Definition 9. Then (C.48) holds.*

$$\Sigma_n^t \models l \Rightarrow \Sigma_{n+1}^t \models l \tag{C.48}$$

with $0 \leq t \leq n$.

Proof: For brevity we only consider positive literals, i.e. $l = f$. We make a case distinction concerning sensing and non-sensing actions:

1. If a_{n+1} is a non-sensing action then transition function (24) evaluates as follows: $\Phi(a_{n+1}, \langle u_n, \Sigma_n \rangle) = \langle u_{n+1}, \Sigma_{n+1} \rangle$ with $u_{n+1} = Res(a_{n+1}, u_n)$ and $\Sigma_{n+1} = \{Res(a_{n+1}, s_n) | s_n \in \Sigma_n\}$. With this and Definition 8 about re-evaluated initial k-states we conclude that $\Sigma_{n+1}^0 = \Sigma_n^0$. Hence, by Definition 9 about re-evaluated c-states it must be true that for an arbitrary f: If $\forall s \in \Sigma_n^t : f \in s$ then $\forall s \in \Sigma_{n+1}^t : f \in s$ and the Lemma is proven for the case of non-sensing actions.

2. If a_{n+1} is a sensing action then transition function (26) evaluates as $\Phi(a_{n+1}, \langle u_n, \Sigma_n \rangle) = \langle u_n, \{s_n | (s_n \in \Sigma_n) \wedge (f \in s_n \Leftrightarrow f \in u_n)\} \rangle$. For this reason $\Sigma_n \supseteq \Sigma_{n+1}$. Hence by Definition 8 it must be true that $\Sigma_n^0 \supseteq \Sigma_{n+1}^0$ and by Definition 9 it must be true that for an arbitrary f: If $\forall s \in \Sigma_n^t : f \in s$ then $\forall s \in \Sigma_{n+1}^t : f \in s$. The Lemma is proven for the case of sensing actions.

Items 1. and 2. prove the Lemma □

Lemma 12 (Re-evaluation does not affect knowledge about the presence). *Given a domain $\mathcal{D}$, a valid initial c-state $\delta_0 = \langle u_0, \Sigma_0 \rangle$ and a sequence of actions $\alpha_n = [a_1; \ldots; a_n]$ such that $\langle u_n, \Sigma_n \rangle = \Phi(a_n, \Phi(a_{n-1}, \cdots \Phi(a_1, \langle u_0, \Sigma_0 \rangle)))$. Let $\Sigma_n^n = Res(a_n, Res(a_{n-1}, \cdots Res(a_1, \Sigma_n^0)))$ be a re-evaluated k-state with Σ_n^0 as the re-evaluated initial state according to Definition 8. Then (C.49) holds.*

$$\Sigma_n = \Sigma_n^n \tag{C.49}$$

Proof:
Induction over n. We show that given (C.49) holds (C.50) holds as well.

$$\Sigma_{n+1} = \Sigma_{n+1}^{n+1} \tag{C.50}$$

where $\langle u_{n+1}, \Sigma_{n+1} \rangle = \Phi(a_{n+1}, \Phi(a_n, \cdots \Phi(a_1, \langle u_0, \Sigma_0 \rangle)))$ is a c-state resulting from the application of the transition functions (24), (26) and $\Sigma_{n+1}^{n+1} = Res(a_{n+1}, Res(a_n, \cdots Res(a_1, \Sigma_{n+1}^0))$ is a re-evaluated k-state with Σ_{n+1}^0 as the re-evaluated initial state according to Definition 8.
Base Step: $\Sigma_0 = \Sigma_0^0$. This emerges from Definitions 2, 8 and 9. (Intuitively, if no action is applied then re-evaluation is not applicable.)
Induction Step: Given that (C.49) holds for one $n \geq 0$, then it holds that $\Sigma_{n+1} = \Sigma_{n+1}^{n+1}$. The re-evaluated c-state after $n+1$ actions is obtained with:

$$\Sigma_{n+1}^{n+1} = \bigcup_{s \in \Sigma_{n+1}^0} Res(a_{n+1}, Res(a_n, \cdots Res(a_1, s))) \tag{C.51}$$

We distinguish whether a_{n+1} is a sensing or non-sensing action:

1. a_{n+1} is a non-sensing action. In this case, according to the transition function (24) it holds that $\forall s : (s \in \Sigma_n \Leftrightarrow Res(a_{n+1}, s) \in \Sigma_{n+1})$. By definition of re-evaluated initial k-states (29) it follows that (C.52) holds.

$$\Sigma_n^0 = \Sigma_{n+1}^0 \tag{C.52}$$

Substituting (C.52) in (C.51) yields:

$$\Sigma_{n+1}^{n+1} = \bigcup_{s \in \Sigma_n^0} Res(a_{n+1}, Res(a_n, \cdots Res(a_1, s))) \tag{C.53}$$

By definition of re-evaluated k-states (29) it holds that $\Sigma_n^n = \bigcup_{s \in \Sigma_n^0} Res(a_n, \cdots Res(a_1, s))$ and we can rewrite (C.53) as:

$$\Sigma_{n+1}^{n+1} = \bigcup_{s \in \Sigma_n^n} Res(a_{n+1}, s) \tag{C.54}$$

By induction hypothesis we can substitute Σ_n^n with Σ_n and have:

$$\Sigma_{n+1}^{n+1} = \bigcup_{s \in \Sigma_n} Res(a_{n+1}, s) \tag{C.55}$$

We reformulate (C.55) as follows:

$$\Sigma_{n+1}^{n+1} = \{Res(a_{n+1}, s) | s \in \Sigma_n\} \tag{C.56}$$

The transition function (24) for non-sensing actions is:

$$\begin{aligned}\Phi(a_{n+1}, \langle u_n, \Sigma_n \rangle) &= \langle u_{n+1}, \Sigma_{n+1} \rangle \\ &= \langle Res(a_{n+1}, u_n), \{Res(a_{n+1}, s) | s \in \Sigma_n\} \rangle\end{aligned}$$

It must therefore hold that

$$\Sigma_{n+1} = \{Res(a_{n+1}, s) | s \in \Sigma_n\} \tag{C.57}$$

It follows from (C.56) and (C.57) that $\Sigma_{n+1}^{n+1} = \Sigma_{n+1}$.

2. a_{n+1} is a sensing action with an arbitrary knowledge proposition $\mathcal{KP}^{a_{n+1}} = f^s$. We make another case distinction:

   (a) $f^s \in u_n$:

   According to 29) the re-evaluated k-state Σ_{n+1}^0 is:

   $$\Sigma_{n+1}^0 = \{s \in \Sigma_0 | Res(a_{n+1}, Res(a_n, \cdots Res(a_1, s))) \in \Sigma_{n+1}\} \tag{C.58}$$

   If a_{n+1} is a sensing action, then $Res(a_{n+1}, s) = s$, and hence:

   $$\Sigma_{n+1}^0 = \{s \in \Sigma_0 | Res(a_n, \cdots Res(a_1, s)) \in \Sigma_{n+1}\} \tag{C.59}$$

   Given that a_{n+1} has a the knowledge proposition $\mathcal{KP}^{a_{n+1}} = f^s$, and $f^s \in u_n$, then from transition function (26) we can conclude that:

   $$\Sigma_{n+1} = \{s \in \Sigma_n | f^s \in s\} \tag{C.60}$$

   Substituting (C.60) in (C.59) yields:

   $$\Sigma_{n+1}^0 = \{s \in \Sigma_0 | Res(a_n, \cdots Res(a_1, s)) \in \{s' \in \Sigma_n | f^s \in s'\}\} \tag{C.61}$$

   By Definition 8, the re-evaluated k-state Σ_n^0 is:

   $$\Sigma_n^0 = \{s \in \Sigma_0 | Res(a_n, \cdots Res(a_1, s)) \in \Sigma_n\} \tag{C.62}$$

   With (C.61) and (C.62) we can conclude that:

   $$\Sigma_{n+1}^0 = \{s \in \Sigma_n^0 | f^s \in Res(a_n, \cdots Res(a_1, s))\} \tag{C.63}$$

   Substituting (C.63) in (C.51) yields:

   $$\Sigma_{n+1}^{n+1} = \bigcup_{s \in \{s_0 \in \Sigma_n^0 | f^s \in Res(a_n, \cdots Res(a_1, s_0))\}} Res(a_{n+1}, Res(a_n, \cdots Res(a_1, s))) \tag{C.64}$$

   For a sensing actions a, it holds that $Res(a, s) = s$ for an arbitrary state s. Hence we write:

   $$\Sigma_{n+1}^{n+1} = \bigcup_{s \in \{s_0 \in \Sigma_n^0 | f^s \in Res(a_n, \cdots Res(a_1, s_0))\}} Res(a_n, \cdots Res(a_1, s)) \tag{C.65}$$

We rewrite (C.65) and separate the union operator as follows:

$$\begin{aligned}\Sigma_{n+1}^{n+1} = &\bigcup_{s\in\Sigma_n^0} Res(a_n, \cdots Res(a_1, s)) \\ &\setminus \bigcup_{s\in\{s_0\in\Sigma_n^0 | f^s \notin Res(a_n,\cdots Res(a_1,s_0))\}} Res(a_n, \cdots Res(a_1, s))\end{aligned} \tag{C.66}$$

With the re-evaluation function (28) and the induction hypothesis (C.49) we have that

$$\begin{aligned}&\bigcup_{s\in\{s_0\in\Sigma_n^0 | f^s \notin Res(a_n,\cdots Res(a_1,s_0))\}} Res(a_n, \cdots Res(a_1, s)) \\ =&\{Res(a_n, \cdots Res(a_1, s)) | s \in \Sigma_n^0 \wedge f^s \notin Res(a_n, \cdots Res(a_1, s))\} \\ \overset{(28)}{=}&\{s \in \Sigma_n^n | f^s \notin s\} \\ \overset{(C.49)}{=}&\{s \in \Sigma_n | f^s \notin s\}\end{aligned} \tag{C.67}$$

Thus, we can rewrite (C.66) as:

$$\Sigma_{n+1}^{n+1} = \bigcup_{s\in\Sigma_n^0} Res(a_n, \cdots Res(a_1, s)) \setminus \{s \in \Sigma_n | f^s \notin s\} \tag{C.68}$$

With the definition of re-evaluated c-states (28) it holds that $\Sigma_n^n = \bigcup_{s\in\Sigma_n^0} Res(a_n, \cdots Res(a_1, s))$ and we can rewrite (C.68) as:

$$\Sigma_{n+1}^{n+1} = \Sigma_n^n \setminus \{s \in \Sigma_n | f^s \notin s\} \tag{C.69}$$

By induction hypothesis we substitute Σ_n^n with Σ_n and obtain:

$$\Sigma_{n+1}^{n+1} = \Sigma_n \setminus \{s \in \Sigma_n | f^s \notin s\} \tag{C.70}$$

Given that $f^s \in u_n$, then the transition function 26 for sensing actions is:

$$\begin{aligned}\Phi(a_{n+1}, \langle u_n, \Sigma_n\rangle) &= \langle u_{n+1}, \Sigma_{n+1}\rangle \\ &= \langle u_n, \{s \in \Sigma_n | f^s \in s\}\rangle\end{aligned} \tag{C.71}$$

Extracting Σ_{n+1} from (C.71) yields:

$$\Sigma_{n+1} = \{s \in \Sigma_n | f^s \in s\} \tag{C.72}$$

We rewrite this as:

$$\Sigma_{n+1} = \Sigma_n \setminus \{s \in \Sigma_n | f^s \notin s\} \tag{C.73}$$

And substitute (C.73) in (C.70) to obtain:

$$\Sigma_{n+1}^{n+1} = \Sigma_{n+1} \tag{C.74}$$

(b) $f^s \notin u_n$: Similar to the case where $f^s \in u_n$.

We have shown that for both sensing and non-sensing actions the induction step holds. This proves the Lemma. □