Approximate Postdictive Reasoning with Answer Set Programming

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

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

Abstract

We present an answer set programming realization of the h-approximation ($\mathcal{HPX}$) theory [8] as an efficient and provably sound reasoning method for epistemic planning and projection problems that involve postdictive reasoning. The efficiency of $\mathcal{HPX}$ stems from an approximate knowledge state representation that involves only a linear number of state variables, as compared to an exponential number for theories that utilize a possible-worlds based semantics. This causes a relatively low computational complexity, i.e, the planning problem is in NP under reasonable restrictions, at the cost that $\mathcal{HPX}$ is incomplete. In this paper, we use the implementation of $\mathcal{HPX}$ to investigate the incompleteness issue and present an empirical evaluation of the solvable fragment and its performance. We find that the solvable fragment of $\mathcal{HPX}$ is indeed reasonable and fairly large: in average about 85% of the considered projection problem instances can be solved, compared to a $\mathcal{PWS}$-based approach with exponential complexity as baseline. In addition to the empirical results, we demonstrate the manner in which $\mathcal{HPX}$ can be applied in a real robotic control task within a smart home, where our scenario illustrates the usefulness of postdictive reasoning to achieve error-tolerance by abnormality detection in a high-level decision-making task.

Keywords: *commonsense reasoning, action and change, epistemic reasoning, answer set programming*

1. Introduction

Starting with the seminal work by [20], a huge body of work concerning logical formalisations of reasoning about action, change and knowledge has been developed (e.g. [23, 5, 6, 15]). In this field, we are particularly interested in the practical application of *epistemic* action theories, e.g. [21, 26, 25], which are concerned with reasoning about the *knowledge* of an agent.
Our existing work [8] focuses in particular on postdictive inference in epistemic action theory. It shows that it is possible to solve the projection problem in the context of reasoning about actions and knowledge in polynomial time, whilst allowing for elaboration tolerant postdictive reasoning. The planning problem is solvable in NP.
In [8], we describe a theoretical transition-function based approach to model the postdictive $\mathcal{HPX}$ theory. However, we left open the question how to implement our theory, and we also did not account for the usability of the theory in terms of its solvable fragment and its actual application in practice. In this paper, we fill this gap and develop a formalisation of $\mathcal{HPX}$ in *answer set programming* (ASP) [2]. We use this implementation to investigate the solvable fragment of the theory and to perform experiments with real robots.

Postdictive Reasoning

As stated in [8], "[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."
An illustrative example is the Litmus test, which was originally introduced in the context of postdictive reasoning in Moore [21]. It illustrates how postdiction determines a world property, which is not directly perceivable, by observing another world property which is a causal consequence of the imperceivable world property. In the litmus example, the the acidic-ness of a liquid is determined by observing the color of the paper.

Example 1. The litmus test [8]
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.

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

In this paper, we take a particular look at using postdiction for abnormality detection in the application domain of robotic environments. For example, a robot can postdict that a door is closed when it tries to pass the door, but its location sensors tell that it does actually not arrive behind the door. However, as we emphasise in Sec. 5, robotics is only one example domain where postdiction is important.

Contributions: Implementation and Application of a Postdictive Epistemic Action Theory

The key contributions emanating from the research presented in this paper are (C1–C2):

C1. Implementation of $\mathcal{HPX}$ within the framework of answer set programming.

C2. Empirical evaluation and application of $\mathcal{HPX}$ in a robotics-based smart home environment.

C1. Formalisation of $\mathcal{HPX}$ as an Answer Set Program

The transition function semantics described in [8] provides a clear formal view on the $\mathcal{HPX}$ theory, but it does not provide an actual implementation which is necessary for an empirical evaluation and the application in real robotic environments. To address this, we present $\mathcal{HPX}$ in terms of answer set programming (ASP). The ASP formalization captures the theory in a model-theoretic form, similar to logic programming implementations of the action language $\mathcal{A}$ [13] or the Discrete Event Calculus (DEC) [22]. The well-understood stable model semantics that underlies ASP makes it possible to show that our implementation is actually sound wrt. the operational semantics (Theorem 1). Consequently, the implementation shares all properties of the operational semantics that are mentioned in [8], i.e. native postdiction, linear number of state variables, temporal knowledge, and concurrent acting and sensing.
Our ASP formalization is implemented a set of translation rules and a set of domain independent logic programming rules. The translation rules compile a domain description specified in an action language syntax into a a set of domain specific logic programming rules (denoted Γ_{world}), which are then combined with the domain independent part (denoted Γ_{hpx}). The domain independent part is a fixed kernel that covers inertia, inference mechanisms and concurrency, as well as plan generation and verification. The domain specific part is generated by our translation rules and covers causation, postdiction and initial knowledge, according to the action language domain specification.
The logic program is processed by an off-the-shelf ASP solver which generates stable models (SM). We show how the SM can be interpreted as conditional plans for applications in cognitive robotics. The same ASP-based framework can also be used for abductive explanation or deductive reasoning in applications like narrative interpretation, smart homes, ambient intelligence, etc.

C2. Empirical Evaluation of the Solvable Fragment of $\mathcal{HPX}$, and its Application in a Smart Home

Section 4 provides a proof of concept of our theory and motivates the approach with a real robotic application. It also contains an empirical evaluation of the theory. Specifically, Section 4.1 investigates (i) the computational performance and (ii) the solvable fragment of our theory. For (i), we measure the computation time required to solve a number of problem instances, and for (ii) we count the number of fluents that are known after a course of actions, and compare it to a $\mathcal{PWS}$-based approach.
Section 4.2 demonstrates how the $\mathcal{HPX}$ planning framework is applied in a real cognitive robotics smart home environment. Here, we use our theory to control actuators and sensors in the smart home. This involves most notably an autonomous robotic wheelchair that can drive autonomously by utilizing a waypoint-based navigation module, along with obstacle-avoidance facilities. The smart home also features automatic doors, illumination control and video cameras. In the context of such environments, we use the postdiction capabilities of $\mathcal{HPX}$ for abnormality detection. Here, we refer to an abnormality as a condition under which an action is not successful. For instance, a typical assistance task in the smart home is *(a)* navigate the wheelchair to a person's location *(b)* pick the person up and *(c)* bring her to her destination. However, an abnormality could e.g. be caused by a chair that blocks an automatic door that would normally open if the chair was not there. If we observe that the door did not open, we can postdict that there was an abnormality and the wheelchair has to pick an alternative route (through another door) to reach its destination. We use the smart home setting as running example throughout this paper.

2. Preliminaries

Our implementation employs the non-monotonic declarative framework of answer set programming to realize the practical application and evaluation of the $\mathcal{HPX}$ theory. In the following we give some preliminaries on ASP and action languages. We refer to [2, 10] for more details.

2.1. Answer Set Programming

Answer Set Programming (ASP) is a form of Logic Programming which uses Negation as Failure (NaF) to implement non-monotonic reasoning. ASP is based on the *Stable Model* (SM) semantics [12] which makes ASP fully declarative. ASP solvers take Logic Programs (LPs) as input and employ the Stable Model Semantics to find solutions to Logic Programs which are called *Answer Sets*. In this work, we consider normal Logic Programs P, which are defined by a set of rules of the form

$$h \leftarrow b_1, \ldots, b_n, \mathit{not}\ b_{n+1}, \ldots, \mathit{not}\ b_{n+m}. \quad \text{with } 0 \leq n \leq n+m. \tag{1}$$

We call h heat atoms and b body atoms. Atoms and negated atoms are called fluent literals.
The symbol not denotes *Negation as Failure* (NaF), also known as *default negation* or *weak negation*. In the following, an atom a or the default negation $\mathit{not}\ a$ of an atom is referred to as a *literal*. The Negation as Failure principle emerges from the Stable Model semantics [12], which is based on the Gelfond-Lifschitz reduct [12] of a Logic Program P wrt. a model M. This is described in Definition 1.

Definition 1 (The Gelfond Lischitz Reduct). *Let $\mathit{body}^+(r)$ be the set of positive literals in a rule r of a Logic Program, and $\mathit{body}^-(r)$ the set of negative literals. The Gelfond-Lifschitz reduct (GL-reduct) of a Logic Program P wrt. a model M, denoted by P^M, is defined as:*

$$P^M = \{\mathit{head}(r) \leftarrow \mathit{body}^+(r) | r \in P \wedge \mathit{body}^-(r) \cap M = \emptyset\} \tag{2}$$

The reduct of a program P wrt. a model M, denoted P^M, is obtained as follows: 1) For all atoms $a \in M$ remove every rule which contains $\mathit{not}\ a$ in its body, so that the rule can not trigger its head atom to be true. 2) For the remaining rules remove all negative literals from their bodies. These can be assumed true as their atoms are not in M anyways.
A reduct P^M must always have one unique minimal model $MM(P^M)$, which is said to be a *Stable Model* of P if it is equal to the original model M. The set of Stable Models of a Logic Program is denoted by $SM[P]$.

Several extensions to the Stable Model Semantics were proposed and implemented to increase the expressiveness of answer set programming and to facilitate the Logic Program design. The extensions that are most important wrt. this work are integrity constraints and choice rules.
Integrity constraints allow one to specify a set of atoms which must not be entailed by a stable model. They are rules where the head is empty, i.e. rules of the following form:

$$\leftarrow b_1, \ldots, b_n, \mathit{not}\ b_{n+1}, \ldots, \mathit{not}\ b_{n+m} \tag{3}$$

Intuitively, the above integrity constraint forbids all Stable Models which contain atoms $b_1, \ldots, b_n$ and which do not contain $b_{n+1}, \ldots, b_m$. For details we refer to literature [10, 2].
Choice Rules are used in the so-called *generation part* of a Logic Program (see e.g. [10]). Intuitively, Choice Rules propose candidate sets of atoms which “generate” a Stable Model if they are compatible with the other rules and constraints in the Logic Program and if they result in a model that is *stable*. Choice Rules are constructs of the form (4).

$$\{h_1, \ldots, h_n\} \leftarrow b_1, \ldots, b_m, \mathit{not}\ b_{m+1}, \mathit{not}\ b_{m+k}. \tag{4}$$

Intuitively, the above rule states that, if the body of rule (4) is entaild in the LP, one stable model is generated for each subset of $\{h_1, \ldots, h_n\}$, which contains the atoms of that subset. For other syntactic sugar, such as cardinality constraints and conditions in ASP we refer the reader to literature [10, 2].
Answer set programming is commonly used to implement action languages, such as the languages $\mathcal{A}$,$\mathcal{B}$,$\mathcal{C}$ [14, 1, 17], or the Event Calculus and the Situation Calculus [18]. Here, non-monotonicity is used to model inertia and the frame problem [24].

2.2. Action Language Syntax and Semantics

Domain specifications in our action language consist of the following elements (1a) – (1d), as described in [8].

$$\textbf{initially}(l_1^{init}, \ldots, l_{n_{init}}^{init}) \tag{5a}$$
$$\textbf{causes}\big(a, l^e, l_1^c \ldots l_{n_c}^c\big) \tag{5b}$$
$$\textbf{determines}(a, f^s) \tag{5c}$$
$$\textbf{executable}\big(a, l_1^{ex} \ldots l_{n_{ex}}^{ex}\big) \tag{5d}$$

(5a). A set of *value propositions* (VPs) denotes initial knowledge. We define $\mathcal{VP}$ as the set of expressions (5a). We sometimes write **initially**$\big(l_1^{init}, \ldots, l_{n_{init}}^{init}\big)$ to denote a set of n_{init} VPs for the respective literals $l_1^{init}, \ldots, l_{n_{init}}^{init}$.

(5b). A set of *effect propositions* (EPs) represent conditional action effects . We call $l^c_1 \dots l^c_{n_c}$ condition literals and l^e an effect literal. For an effect proposition ep of the form (5b) 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.

(5c). *Knowledge propositions* (KPs) of the form (5c) 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.

(5d). *Executability conditions* (EXCs) of the form (5d) 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 **executable**$(a, \emptyset)$ is implicit for any action a without an explicit executability condition.

The semantics of action languages is usually defined in terms of a transition function that maps actions and states to states, as, e.g., described in [14]. States are represented as sets of fluents. For example, given an effect proposition **causes** (a, l^e, l^c), a transition function ϕ in the action language $\mathcal{A}$ [13] maps the action a, together with any state that contains the fluent literal l^c, to a state which contains l^e. In other words, l^e holds after a was applied on a state in which l^c holds. The most common problems that action languages solve are the projection problem and the planning problem. The projection problem is that of determining whether a certain literal l holds after a given sequence of state transitions in a given initial state. The planning problem is that of determining a sequence of state transitions to obtain a certain goal literal l given an initial state. For details we refer to [8].

3. Answer Set Programming Formalization of $\mathcal{HPX}$

To make $\mathcal{HPX}$ applicable in practice, we implement the theory in terms of answer set programming and realize the core $\mathcal{HPX}$ inference mechanisms for causation, forward and backward inertia, as well as positive and negative postdiction presented in [8, Sec. 3] as logic programming rules. In addition, we specify a set of rules for epistemic action planning. Section 3.1 describes the main predicates and how the temporal dimension of knowledge is represented. An $\mathcal{HPX}$-logic program consists of a domain-specific and a domain-independent part (see Section 3.2). The domain-specific part is generated by seven *translation rules* (T1) – (T7), that compile the action language syntax into LP rules (see Section 3.3). The domain-independent part is a fixed set rules to model inertia and sensing (see Section 3.4). The stable models that result from the logic programs are interpreted as conditional plans, in order to realize decision making tasks for cognitive robotics and other application domains (see Section 3.5). The logic programming implementation of $\mathcal{HPX}$ can be understood as an alternative model-theoretic semantics of $\mathcal{HPX}$. The relation between the model-theoretic semantics and the operational $\mathcal{HPX}$ semantics is illustrated in Section 3.6, which also contains a corresponding soundness theorem.

3.1. Main Predicates and Notation

The following are the main predicates used in the ASP formalization:

- $knows(l, t, n, b)$ states that at step n in branch b it is known that l holds (or did hold) at step t (with $t \leq n$).
- $occ(a, n, b)$ denotes that action a occurs at step n in branch b.
- $apply(ep, n, b)$ denotes that an effect proposition ep is applied at step n in branch b. Whenever $occ(a, n, b)$ and ep is an effect proposition of a, then $apply(ep, n, b)$. This reflects the abstraction from action histories to effect histories in [8, Def. 1].
- $sRes(l, n, b, b')$ denotes that the literal l is sensed at step n in branch b, such that it will hold in the child branch b'.
- $uBr(n, b)$ denotes that branch b is a valid branch at step n. Actions can only be executed if a branch is valid.

As a notational convention, for negative literals we write $\neg f$ to denote $neg(f)$ in ASP syntax.

3.2. Constitution of an $\mathcal{HPX}$-logic program

The formalization is based on a domain independent foundational theory Γ_{hpx} and on a set of *translation rules* **T** that are applied to a planning domain $\mathcal{D}$. An ASP formalization of $\mathcal{D}$, denoted by LP($\mathcal{D}$), consists of a domain dependent theory and a domain independent theory:

- Domain dependent theory (Γ_{world}): It consists of a set of rules $\Gamma_{fluents}$ representing fluent declarations, Γ_{init} representing initial knowledge; Γ_{act} representing actions; and Γ_{goal} representing goals.[1]

[1]We use the term *goals* in the context of action planning. However, one could similarly use the term *query* to emphasize that such statements query whether a fluent literal is known to hold after the execution of a plan.

- Domain independent theory (Γ_{hpx}): This consists of a set of auxiliary definitions Γ_{aux}; a set of rules to handle inertia (Γ_{in}); sensing (Γ_{sen}); inference mechanisms (Γ_{infer}); concurrency (Γ_{conc}); plan verification (Γ_{verify}) and plan-generation & optimization (Γ_{plan}).

The resulting logic program, denoted LP($\mathcal{D}$), is given as:

$$LP(\mathcal{D}) = \underbrace{\left[\,\Gamma_{aux} \cup \Gamma_{in} \cup \Gamma_{sen} \cup \Gamma_{infer} \cup \Gamma_{conc} \cup \Gamma_{verify} \cup \Gamma_{plan}\,\right]}_{\Gamma_{hpx}} \cup \underbrace{\left[\,\Gamma_{fluents} \cup \Gamma_{init} \cup \Gamma_{act} \cup \Gamma_{goal}\,\right]}_{\Gamma_{world}} \tag{6}$$

We call a logic program that is assembled according to (6) an $\mathcal{HPX}$-logic program for a domain specification $\mathcal{D}$.

3.3. *Translation Rules:* $(\mathcal{D} \xrightarrow{T1-T7} \Gamma_{world})$

The domain dependent theory Γ_{world} is obtained by applying the set of translation rules $\mathbf{T} = \{T1, \ldots, T7\}$ on a planning domain $\mathcal{D}$, specified in the action language syntax. Translation rule T1 declares fluents by generating $\Gamma_{fluents}$, rule T2 generates Γ_{init}, rules T3 – T6 generate Γ_{act} and rule T7 generates Γ_{goal}.

3.3.1. *Fluent Declarations* $(\mathcal{D} \xrightarrow{T1} \Gamma_{fluents})$

For every fluent $f \in \mathcal{F}_{\mathcal{D}}$, LP($\mathcal{D}$) contains a fact (T1).

$$fluent(f) \tag{T1}$$

3.3.2. *Initial Knowledge* $(\mathcal{VP} \xrightarrow{T2} \Gamma_{init})$

Facts Γ_{init} for initial knowledge are obtained by applying translation rule (T2). For each value proposition **initially**(l^{init}) (1a), we generate the fact:

$$knows(l^{init}, 0, 0, 0) \tag{T2}$$

3.3.3. *Actions* $(\mathcal{EP}, \mathcal{KP}, \mathcal{EXC} \xrightarrow{T3–T6} \Gamma_{act})$

The logic programming rules that represent actions involve effect propositions, knowledge propositions and executability conditions. They are generated by translation rules T3 – T6, and constitute the main part of the domain-specific part of an $\mathcal{HPX}$ logic program.

Declaration of Effect Propositions.

For every effect proposition (1b) of the form **causes** $\left(a, l^e, \{l^c_1 \ldots l^c_{n_c}\}\right)$, LP($\mathcal{D}$) contains (T3). Here, *ep* is an automatically generated identifier for the particular EP of a, $hasCond$ represents condition literals, $hasEff$ represents effect literals and $hasEP$ assigns an effect proposition.

$$\begin{aligned} &hasEP(a, ep) \\ &hasEff(ep, l^e) \\ &hasCond(ep, l^c_1) \quad \ldots \quad hasCond(ep, l^c_{n_c}) \end{aligned} \tag{T3}$$

Knowledge Level Effects of Effect Propositions.

Knowledge-level effects of EP represent knowledge gain by causation and postdiction, i.e. they reflect inference mechanisms **IM.3 – IM.5** as specified in the operational $\mathcal{HPX}$ semantics [8].[2]

$$kCause(l^e, T+1, N, B) \leftarrow apply(ep, T, B), N > T, knows(l^c_1, T, N, B), \ldots, knows(l^c_k, T, N, B) \tag{T4a}$$

$$kPosPost(l^c_i, T, N, B) \leftarrow apply(ep, T, B), knows(l^e, T+1, N, B), knows(\overline{l^e}, T, N, B) \tag{T4b}$$

$$kNegPost(\overline{l^{c-}_i}, T, N, B) \leftarrow apply(ep, T, B), knows(\overline{l^e}, T+1, N, B), knows(l^{c+}_{i_1}, T, N, B), \ldots, knows(l^{c+}_{i_k}, T, N, B) \tag{T4c}$$

► *Causation* (T4a). This refers to inference mechanism **IM.3** in [8, Eq. (11)]. After an arbitrary number of steps n, if all condition literals l^c_i of an EP (1b) are known to hold at t, and if the action is applied at t, then at step $n > t$, it is known that its effect l^e holds at $t+1$. This is denoted by the atom $kCause(l^e, t+1, n, b)$ which reads as "by causation inference it is known that at step n in branch b literal l^e is known to hold at step $t+1$". The atom apply(ep,t,b) represents that action a with the EP ep happens at t in b.

► *Positive postdiction* (T4b). In the operational semantics, positive postdiction is defined as inference mechanism **IM.4** in [8, Eq. (12)]. In the ASP formulation, we iterate over condition literals $l^c_i \in \{l^c_1, \ldots, l^c_{n_c}\}$ of an effect proposition ep and add a rule (T4b) to the LP for each condition literal l^c_i. This defines how knowledge about the condition of an effect proposition is postdicted by knowing that the effect holds after the action but did not hold before. For example, if at n in b it is known that the complement $\overline{l^e}$ of an effect literal of an EP holds (i.e., $knows(\overline{l^e}, t, n, b)$), and if the EP is applied at t, and if it is known that the effect literal holds at $t+1$ ($knows(l^e, t+1, n, b)$), then the application of the EP must have produced the effect. Therefore one can conclude

[2]The frame problem is handled due to Negation as Failure that inheres in the stable model semantics (see e.g. [18]).

that the conditions $\{l_1^c, \ldots, l_{n_c}^c\}$ of the EP must hold at t. This is represented by the atom $kPosPost(\overline{l_i^{c-}}, t, n, b)$ which reads as "by positive postdiction it is known that at step n in branch b literal $\overline{l_i^{c-}}$ is known to hold at step t".

▶ *Negative postdiction* (T4c). Negative Postdiction is defined as inference mechanism **IM.5** in [8, Eq. (13)]. We iterate over each potentially unknown condition literal $l_i^{c-} \in \{l_1^c, \ldots, l_{nc}^c\}$ of an effect proposition ep. For each literal l_i^{c-}, we add one rule (T4c) to the program, where $\{l_{i_1}^{c+}, \ldots, l_{i_{nc}}^{c+}\} = \{l_1^c, \ldots, l_{nc}^c\} \backslash l_i^{c-}$ are the condition literals that are known to hold. An atom $kNegPost(\overline{l_i^{c-}}, t, n, b)$ denotes that "by negative postdiction it is known that at step n in branch b literal $\overline{l_i^{c-}}$ is known to hold at step t". This covers the case where we postdict that a condition must be false if the effect is *known not to hold* after the action and all other conditions are known to hold. For example, if at n it is known that the complement of an effect literal l holds at some $t+1$, and if the EP is applied at t, and if it is known that all condition literals hold at t, except one literal l_i^{c-} for which it is unknown whether it holds. Then the complement of l_i^{c-} must hold because otherwise the effect literal would hold at $t+1$. Note that this only works correctly if concurrent effect porpositions have different effect literals (see rule (F2b)).

Knowledge Propositions.
We assign a knowledge proposition (KP) (1c) to an action a using a $hasKP$ predicate:

$$hasKP(a, f) \tag{T5}$$

Executability Conditions.
These reflect what an agent must know to execute an action. For each executability condition **executable** $\left(a, \{l_1^{ex} \ldots l_{n_{ex}}^{ex}\}\right)$ (1d), LP($\mathcal{D}$) contains the following constraints:

$$\begin{array}{ll} \leftarrow occ(a, N, B), not\ knows(l_1^{ex}, N, N, B) & \ldots \\ \leftarrow occ(a, N, B), not\ knows(l_{n_{ex}}^{ex}, N, N, B) & \end{array} \tag{T6}$$

Example 2 demonstrates how translation rules (T3) – (T6) generate the logic programming rules for Γ_{act}.

Example 2. Generating Γ_{act}
Consider an action `drive` with the effect proposition **causes**(`drive`, `in_room`, `is_open`) and the executability condition **executable**(`drive`, `is_open`). The effect propositions is defined by (T3) as follows:

$$hasEP(\mathtt{drive}, \mathtt{ep\_drive\_0}) \qquad hasEff(\mathtt{ep\_drive\_0}, \mathtt{in\_room}) \qquad hasPC(\mathtt{ep\_drive\_0}, \mathtt{open})$$

Here, `ep_drive_0` is a syntactically generated label for the 0-th effect proposition of the `drive` action. The knowledge-level effects of the action are generated through (T4a–T4c):

$$\begin{aligned} kCause(\mathtt{in\_room}, T+1, N, B) &\leftarrow apply(\mathtt{ep\_drive\_0}, T, B), N > T, knows(\mathtt{open}, T, N, B) \\ kPosPost(\mathtt{open}, T, N, B) &\leftarrow apply(\mathtt{ep\_drive\_0}, T, B), knows(\mathtt{in\_room}, T+1, N, B), knows(\neg\mathtt{in\_room}, T, N, B) \\ kNegPost(\neg\mathtt{open}, T, N, B) &\leftarrow apply(\mathtt{ep\_drive\_0}, T, B), knows(\neg\mathtt{in\_room}, T+1, N, B) \end{aligned}$$

The first rule refers to *causation* (T4a): If it is known that the door is open, then it is known that the robot arrives in the living room after driving. The second rule represents *positive postdiction* (T4b): If it is known that the robot arrived in the living room, then it must be true that the door was open while it was driving. The third rule captures *negative postdiction* (T4c): If the robot did not arrive in the living room, then the door must have been closed.

Finally, (T6) generates the executability constraint $\leftarrow occ(\mathtt{drive}, N, B), not\ knows(\mathtt{in\_room}, N, N, B)$

3.3.4. Goals $(\mathcal{G} \xmapsto{T7} \Gamma_{goal})$

Let $\mathcal{G} = \left\langle \{l_1^{sg}, ..., l_{n_{sg}}^{sg}\}, \{l_1^{wg}, ..., l_{n_{wg}}^{wg}\} \right\rangle$ denote a pair of sets of goals. Here, $\{l_1^{sg}, ..., l_{n_{sg}}^{sg}\}$ is a set of literals that represent *strong goals*, i.e. a set of literals that must be *necessarily known to hold* (see Eq. ([8, 18)]), given a domain and a plan. Further, $\{l_1^{wg}, ..., l_{n_{wg}}^{wg}\}$ is a set of literals that represent *weak goals*, i.e. the literals must be *possibly known to hold* (see Eq. ([8, 19)]). With these specifications, the logic program contains facts generated by T7.

$$wGoal(l_1^{wg}) \quad \ldots \quad wGoal(l_{n_{wg}}^{wg}) \tag{T7a}$$

$$sGoal(l_1^{sg}) \quad \ldots \quad sGoal(l_{n_{sg}}^{sg}) \tag{T7b}$$

These facts are used to trigger integrity constraints (F6a) and (F6d) of the domain independent theory that rule out those stable models where the goals are not achieved.

3.4. Γ_{hpx} – Foundational Theory (F1) – (F7)

The foundational domain independent $\mathcal{HPX}$-theory covers auxiliaries (F1), concurrency (F2), inertia (F3), inference mechanisms (F4), sensing (F5), plan verification (F6) as well as plan generation and optimization (F7).

F1. Preliminaries and Auxiliary Definitions (Γ_{aux}) First, the maximal plan length and with is defined by instantiating atoms for steps (s) and branches (br) (F1a). Here, `maxS` and `maxBr` are constants of which the value is passed to the logic program at execution time. To speed up the solving process we precompute inequality of branch labels with (F1b).

$$s(0..\texttt{maxS}) \qquad br(0..\texttt{maxBr}) \tag{F1a}$$

$$neq(B_1, B_2) \leftarrow B_1 \neq B_2, br(B_1), br(B_2) \tag{F1b}$$

In (F1c) we declare fluents f and their negations $neg(f)$ as literals, and we define the complement of literals.[3]

$$\begin{array}{rr} literal(neg(F)) \leftarrow fluent(F) & literal(F) \leftarrow fluent(F) \\ complement(neg(F), F) \leftarrow fluent(F) & complement(L_1, L_2) \leftarrow complement(L_2, L_1) \end{array} \tag{F1c}$$

F2. Concurrency (Γ_{conc}) Concurrency and the abstraction of effect propositions from actions is implemented as follows:

$$apply(EP, N, B) \leftarrow hasEP(A, EP), occ(A, N, B) \tag{F2a}$$

$$\leftarrow apply(EP_1, T, B), hasEff(EP_1, L), apply(EP_2, T, B), hasEff(EP_2, L), EP_1 \neq EP_2, br(B), literal(L) \tag{F2b}$$

Rule (F2a) applies all effect propositions of an action a if that action occurs. (F2b) is a restriction concerning the application of similar effect propositions: two effect propositions are similar if they have the same effect literal. The restriction is necessary, because otherwise the positive postdiction rule (T4b) and the inertia law (F3) would not work correctly. An explanation is given with the corresponding rules F3, T4b.

F3. Inertia (Γ_{in}) Inertia is applied in both forward and backward direction similar to [13]. To formalize this, we need a notion on knowing that a literal is *not* set by an action. This is expressed with the predicate $kNotSet$.

$$kNotSet(L, T, N, B) \leftarrow not\ kMaySet(L, T, B), uBr(N, B), s(T), literal(L) \tag{F3a}$$

$$kMaySet(L, T, B) \leftarrow apply(EP, T, B), hasEff(EP, L) \tag{F3b}$$

$$\begin{aligned} kNotSet(L, T, N, B) \leftarrow & apply(EP, T, B), hasCond(EP, L'), hasEff(EP, L), \\ & knows(\overline{L'}, T, N, B), complement(L', \overline{L'}), N >= T. \end{aligned} \tag{F3c}$$

A literal could be known to be not set for two reasons: 1) if no effect proposition with the respective effect literal is applied, then this fluent can not be initiated. $kMaySet(l, t, b)$ (F3b) represents that at t an EP with the effect literal l is applied in branch b. If $kMaySet(l, t, b)$ does not hold then l is known not to be set at t in b (F3a). 2) a literal is known not to be set if an effect proposition with that literal is applied, but one of its conditions is known not to hold (F3c). Note that this requires the concurrency restriction (F2b), because without that restriction a literal could still be set by another effect proposition. We can now formulate forward inertia (F3d) and backward inertia (F3e) as follows. Atoms $kFwdInertia/4$ represent knowledge gained by forward inertia and atoms $kBackInertia/4$ represent knowledge gained by backward inertia.

$$kFwdInertia(L, T, N, B) \leftarrow knows(L, T-1, N, B) kNotSet(\overline{L}, T-1, N, B), complement(L, \overline{L}), T \leq N \tag{F3d}$$

$$kBackInertia(L, T, N, B) \leftarrow knows(L, T+1, N, B), kNotSet(L, T, N, B), N > T \tag{F3e}$$

Inertia is represented as inference mechanisms **IM.1** (forward inertia) and **IM.2** (backward inertia) in [8, Eqs. (9), (10)]. The above inertia rules refer to a mental operation of inferring the temporal propagation of facts within an agent's knowledge. In addition, the operational semantics of $\mathcal{HPX}$ implies that knowledge itself is inertial (see [8, Lemma 5]). That is, if at n it is known that l holds at t, then this is also known at $n+1$. Inertia of knowledge implemented as follows:

$$knows(L, T, N, B) \leftarrow knows(L, T, N-1, B), N \leq \texttt{maxS} \tag{F3f}$$

F4. Inference mechanisms (Γ_{infer}) The following rules are required to transfer the result of causation and positive and negative postdiction to knowledge.

$$knows(L, T, N, B) \leftarrow kFwdInertia(L, T, N, B) \tag{F4a}$$

$$knows(L, T, N, B) \leftarrow kBackInertia(L, T, N, B) \tag{F4b}$$

$$knows(L, T, N, B) \leftarrow kCause(L, T, N, B) \tag{F4c}$$

$$knows(L, T, N, B) \leftarrow kPosPost(L, T, N, B) \tag{F4d}$$

$$knows(L, T, N, B) \leftarrow kNegPost(L, T, N, B) \tag{F4e}$$

[3]Recall, that we often write $\neg f$ as a shorthand for $neg(f)$ to denote the negation of a fluent.

F5. Sensing and Branching (Γ_{sense}) If sensing occurs, then each possible outcome of the sensing is assigned to a different branch, where $uBr(n, b)$ denotes that branch b is used at step n. Initially, only branch 0 is used (F5a).
$sNextBr$ (F5b) is an auxiliary predicate to denote that sensing produced a child branch. This is used in (F5c), which denotes that if no sensing result was produced, then a branch that was used in the past (at $n-1$) is used now (at n). If sensing did produce a child branch, i.e. if $sNextBr(n, b)$ is triggered, then $uBr(n, b')$ will be set by (F5j), where b' is the child branch's label.

$$uBr(0, 0) \tag{F5a}$$
$$sNextBr(N, B) \leftarrow sRes(L, N, B, B') \tag{F5b}$$
$$uBr(N, B) \leftarrow uBr(N-1, B), not\ sNextBr(N-1, B), s(N) \tag{F5c}$$

An auxiliary predicate kw (F5d),(F5e) is an abbreviation for *knowing whether*:

$$kw(F, T, N, B) \leftarrow knows(F, T, N, B). \tag{F5d}$$
$$kw(F, T, N, B) \leftarrow knows(\neg F, T, N, B). \tag{F5e}$$

Next we describe the key rules that generate the sensing results, i.e. the $sRes$ atoms: atoms $occ(a, n, b), hasKP(a, f)$ denote that a sensing action with the knowledge proposition f occurs at step n in branch b. Sensing generates two branches. The positive sensing result is assigned to the original branch via (F5f). However, the sensing result it generated only if the negative sensing result is not already known to hold.
For the negative result, the choice rule (F5g) "picks" a valid child branch. It must be restricted that two sensing actions which occur at the same step n but in different branches b pick the same child branch (F5h). It must further be restricted that the negative sensing result is not assigned to already used branches (F5i).

$$sRes(F, N, B, B) \leftarrow occ(A, N, B), hasKP(A, F), not\ kw(F, N, N, B) \tag{F5f}$$
$$1\{sRes(\neg F, N, B, B') : neq(B, B')\}1 \leftarrow occ(A, N, B), hasKP(A, F), not\ kw(F, N, N, B) \tag{F5g}$$
$$\leftarrow 2\{sRes(L, N, B, B') : br(B) : literal(L)\}, br(B'), step(N) \tag{F5h}$$
$$\leftarrow sRes(L, N, B, B'), uBr(N, B'), literal(L), neq(B, B') \tag{F5i}$$

If an $sRes$ atom is produced, then the assigned branch is marked as used (F5j). Sensing results affect knowledge through (F5k). Note, that like in the $sense$ function [8, Eq. (7)] of the operational semantics, sensing yields the value of a fluent at the time it was changed, i.e. at $N-1$ and not at N.
Finally, we need to implement the $\mathcal{HPX}$-restriction that two fluents can not be sensed concurrently, because this would cuas an exponential growth of the search tree, as described in [8, Eq. (7)]. This restriction is realized with (F5l).

$$uBr(N, B') \leftarrow sRes(L, N-1, B, B'), s(N). \tag{F5j}$$
$$knows(L, N-1, N, B') \leftarrow sRes(L, N-1, B, B'), s(N). \tag{F5k}$$
$$\leftarrow 2\{occ(A, N, B) : hasKP(A, \_)\}, br(B), s(N). \tag{F5l}$$

In order to to apply postdiction, causation and inertia rules to a child branch resulting from a sensing action, the child branch has to inherit knowledge (F5m) and application of EPs (F5n) from the parent branch.

$$knows(L, T, N, B') \leftarrow sRes(\_, N-1, B, B'), B \neq B', knows(L, T, N-1, B), N \geq T \tag{F5m}$$
$$apply(EP, T, B') \leftarrow sRes(\_, N, B, B'), B \neq B' apply(EP, T, B), N \geq T \tag{F5n}$$

F6. Plan Verification (Γ_{verify}) The ASP formalization supports both weak and strong goals. For weak goals there must exist one leaf where all goal literals are achieved and for strong goals the goal literals must be achieved in all leafs. Weak or strong goals are declared with the $wGoal$ and $sGoal$ predicates and defined through translation rules (T7). (F6a) defines atoms $notWG(n, b)$ which denote that a weak goal is not achieved at step n in branch b. An atom $allWGAchieved(N)$ reflects whether all weak goals are achieved at a step N (F6b). If they are not achieved at step `maxS`, then a corresponding model is not stable (F6c).

$$notWG(N,B) \leftarrow wGoal(L), uBr(N,B), not\ knows(L,N,N,B), literal(L). \tag{F6a}$$
$$allWGAchieved(N) \leftarrow not\ notWG(N,B), uBr(N,B). \tag{F6b}$$
$$\leftarrow not\ allWGAchieved(\texttt{maxS}). \tag{F6c}$$

Similarly, $notSG(n,b)$ denotes that a strong goal is not achived at step n in branch b (F6d). In contrast to weak goals, strong goals must be achieved in all used branches at the final step $\texttt{maxS}$ (F6e).

$$notSG(N,B) \leftarrow sGoal(L), uBr(N,B), not\ knows(L,N,N,B), literal(L) \tag{F6d}$$
$$\leftarrow notSG(\texttt{maxS}, B), uBr(\texttt{maxS}, B) \tag{F6e}$$

Information about nodes where goals are not yet achieved is also generated. This is used in the plan generation part for pruning (F7a)–(F7b).

$$\begin{aligned} notGoal(N,B) &\leftarrow notSG(N,B) \\ notGoal(N,B) &\leftarrow notWG(N,B) \end{aligned} \tag{F6f}$$

F7. Plan Generation and Optimization (Γ_{plan}) In the generation part of the logic program, (F7a) and (F7b) implement sequential and concurrent planning respectively: for concurrent planning the choice rule's upper bound "1" is simply removed.[4] Choice rules are used to "generate" atoms and hence can be interpreted as those mechanisms which span up the search tree (see [10] for details). Optimal plans in terms of the number of actions are generated with the optimization statement (F7c), at the cost that computational complexity of the ASP solving raises from NP-completeness to Δ_2^P-completeness (see e.g. [10]).

$$1\{occ(A,N,B) : a(A)\}1 \leftarrow uBr(N,B), notGoal(N,B), N < \texttt{maxS}. \tag{F7a}$$
$$1\{occ(A,N,B) : a(A)\} \leftarrow uBr(N,B), notGoal(N,B), N < \texttt{maxS}. \tag{F7b}$$
$$\#minimize\{occ(\_,\_,\_)@1\}. \tag{F7c}$$

3.5. *Plan Extraction from stable models*

To formally define how concurrent conditional plans (CCP) relate to stable models, we define a function $trans$ that takes a set of atoms S and two integer numbers $0 \leq n \leq \texttt{maxS}$, $0 \leq b \leq \texttt{maxB}$ as input and produces a CCP as output. n and b describe the position of the plan's root node in the transition tree, i.e. $trans(S,0,0)$ yields the conditional plan starting at the initial state.

$$trans(S,n,b) = \begin{cases} [] & \text{if } n = \texttt{maxS} \\ [[a_1||\ldots||a_m]; trans(S,n+1,b)] & \text{if } \neg\exists b', f : sRes(\neg f,n,b,b') \in S \\ [[a_1||\ldots||a_m]; \texttt{if } \neg f \texttt{ then } trans(S,n+1,b') & \text{if } \exists b', f : sRes(\neg f,n,b,b') \in S \\ \qquad\qquad\qquad\qquad \texttt{else } trans(S,n+1,b)] & \end{cases} \tag{7}$$

where $\{a_1,\ldots,a_m\} = \{a | occ(a,n,b) \in \boldsymbol{P}\}$ and $\texttt{maxS}$ is a constant that limits the plan depth.

3.6. *Relation between the ASP Implementation and the Operational $\mathcal{HPX}$-Semantics*

The translation function (7) relates the occurrence of actions in the ASP implementation of $\mathcal{HPX}$ to state transitions in the operational semantics but it does not guarantee that the epistemic effects of actions are sound. In the following we investigate the relation between the operational semantics and the ASP implementation and state a soundness theorem.

As a prerequisite we define the auxiliary function (8) which describe a parent-child-relation between nodes in the transition tree. Let S be a stable model and $0 \leq n \leq \texttt{maxS}$, $0 \leq b \leq \texttt{maxB}$, $0 \leq b' \leq \texttt{maxB}$ be integers which are used to represent nodes in the transition tree. Then a function which defines whether a branch b' is a child branch of a node $\langle n,b \rangle$ wrt. a set of atoms S is defined as follows:

$$hasChild(n,b,b',S) \begin{cases} true & \text{if } \exists l : sRes(l,n,b,b') \in S \\ true & \text{if } b = b' \wedge \neg\exists l : sRes(l,n,b,b') \in S \\ false & \text{otherwise} \end{cases} \tag{8}$$

This is used to define the following ancestor relation:

$$ancestor(n_1,b_1,n_2,b_2,S) = \begin{cases} true & \text{if } \exists n,b : (ancestor(n_1,b_1,n,b,S) \wedge hasChild(n,b,b_2,S) \wedge n_2 = n+1) \\ false & \text{otherwise} \end{cases} \tag{9}$$

[4]In an actual implementation the LP may of course contain only one of these two choice rules, depending on which kind of planning is desired.

We are now ready to define the following auxiliary functions (10) which are used to map atoms in a stable model to nodes in the transition tree of the operational semantics:

$$\boldsymbol{\kappa}(n,b,S) = \{\langle l,t\rangle \,|\, knows(l,t,n,b) \in S\} \tag{10a}$$
$$\boldsymbol{\alpha}(n,b,S) = \{\langle a,t\rangle \,|\, \exists b',t : (occ(a,t,b') \in S \wedge ancestor(t,b',n,b,S))\} \tag{10b}$$
$$\mathfrak{h}(n,b,S) = \langle \boldsymbol{\alpha}(n,b,S), \boldsymbol{\kappa}(n,b,S)\rangle \tag{10c}$$
$$\boldsymbol{\epsilon}(n,b,S) = \boldsymbol{\epsilon}(\mathfrak{h}(n,b,S)) \tag{10d}$$

We also presume the following Definition 2 as a notational convention to formally describe the relation between the ASP formalization and the operational semantics.

Definition 2 (Notation to relate ASP implementation and $\mathcal{HPX}$ semantics).

- *$\texttt{maxS}$ and $\texttt{maxB}$ are constants denoting the maximal plan depth and width respectively. $0 \leq n \leq \texttt{maxS}$, $0 \leq b \leq \texttt{maxB}$ and $0 \leq b' \leq \texttt{maxB}$ denote variables for steps and branches respectively.*
- *$\mathcal{D}$ is a domain description with the initial h-state $\mathfrak{h}_0$*
- *$LP(\mathcal{D})$ is the logic program of a domain description $\mathcal{D}$ without without the plan-generation rule (F7) and without the goal statements generated by translation rule (T7)*
- *$S_{\mathcal{D}}^{\boldsymbol{P}}$ is a stable model of $LP(\mathcal{D}) \cup \boldsymbol{P}$ where $\boldsymbol{P}$ is a set of $occ(a,n,b)$ atoms with $0 \leq n < \texttt{maxS}$ such that*
  - $\forall a,n,b : \big(occ(a,n,b) \in S_{\mathcal{D}}^{\boldsymbol{P}} \Rightarrow uBr(n,b)\big)$.[5]
  - $\forall n,b : \big(uBr(n,b) \in S_{\mathcal{D}}^{\boldsymbol{P}} \Rightarrow \exists a : occ(a,n,b) \in S_{\mathcal{D}}^{\boldsymbol{P}}\big)$ [6]
- *$\boldsymbol{A}_{n,b} = \{a | occ(a,n,b) \in S_{\mathcal{D}}^{\boldsymbol{P}}\}$ is a set of actions applied at a transition tree node with the "coordinates" $\langle n,b\rangle$.*

Functions (8) and (10) along with Definition 2 allow us to provide a complete summary of how ASP atoms relate to the operational $\mathcal{HPX}$ semantics in Table 1.
The notational conventions allow us to state Theorem 1, which is the core soundness theorem concerning knowledge. It states that if there exists a branch b' where a literal l is known to hold in the ASP formalisation, then there exists a leaf in the operational semantics where l is also known to hold.

Theorem 1 (Soundness for knowledge atoms for single state transitions). *For all n,t,b, if there exists a branch b' such that $hasChild(n,b,b',S_{\mathcal{D}}^{\boldsymbol{P}})$ and $knows(l,t,n{+}1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}}$, then there exists an h-state $\mathfrak{h} \in \Psi(\boldsymbol{A}_{n,b}, \mathfrak{h}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}))$ such that $\mathfrak{h} \models \langle l,t\rangle$. Formally:*

$$\begin{gathered}\forall n,b,b' : \big(hasChild(n,b,b',S_{\mathcal{D}}^{\boldsymbol{P}}) \Rightarrow \\ \big(\exists \mathfrak{h} \in \Psi(\boldsymbol{A}_{n,b}, \mathfrak{h}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}})) : \forall l,t : (knows(l,t,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Rightarrow \mathfrak{h} \models \langle l,t\rangle\,)\big)\end{gathered} \tag{11}$$

Proof sketch: [*Complete proof*: **Appendix B**]
We first make a case distinction concerning whether or not $\exists l' : sRes(l',n,b,b') \in S_{\mathcal{D}}^{\boldsymbol{P}}$. This determines whether $hasChild(n,b,b',S_{\mathcal{D}}^{\boldsymbol{P}})$ is true and eliminates the $\exists$-quantification over $\mathfrak{h}$. Then we perform induction over the structure of the following implications (12) that result from the stable model semantics and our described ASP implementation, and that are exhaustive in producing knowledge about pairs $\langle l,t\rangle$.

$$knows(l,t,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftarrow \Big(t = 0 \wedge n = -1 \wedge b' = 0 \wedge l \in \mathcal{VP}\Big) \tag{12a}$$
$$knows(l,t,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftarrow \big(knows(l,t-1,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \wedge kNotSet(\bar{l},t-1,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \wedge t \leq n+1\big) \tag{12b}$$
$$knows(l,t,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftarrow \big(knows(l,t+1,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \wedge kNotSet(l,t,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \wedge t < n+1\big) \tag{12c}$$
$$knows(l,t,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftarrow knows(l,t,n,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \tag{12d}$$
$$knows(l,t,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftarrow kCause(l,t,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \tag{12e}$$
$$knows(l,t,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftarrow kPosPost(l,t,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \tag{12f}$$
$$knows(l,t,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftarrow kNegPost(l,t,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \tag{12g}$$
$$knows(l,t,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftarrow \big(\exists b : sRes(l,n,b,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \wedge t = n\big) \tag{12h}$$
$$knows(l,t,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftarrow \big(\exists b : (sRes(l',n,b,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \wedge knows(l,t,n,b) \in S_{\mathcal{D}}^{\boldsymbol{P}}) \wedge n \geq t\big) \tag{12i}$$

[5] This restriction reflects the mechanics of the plan generation rule (F7) which only generates $occ(a,n,b)$ atoms if $uBr(n,b) \in S_{\mathcal{D}}^{\boldsymbol{P}}$.
[6] This ensures that there are no "gaps" in a plan, i.e. for all nodes in used branches there occurs at least one action. Note that this restriction is met for all $occ/3$ atoms which are generated by the plan generation rule (F7).

	ASP formalization	**Operational semantics**	
Core predicates			
Knowledge	$knows(l, t, n+1, b') \in S_{\mathcal{D}}^{\mathbf{P}}$	$\langle l, t\rangle \in \boldsymbol{\kappa}(\mathfrak{h})$	Theorem 1
Inertial fluents	$kNotSet(\bar{l}, t, n+1, b') \in S_{\mathcal{D}}^{\mathbf{P}}$	$inertial(l, t, \mathfrak{h})$	Lemma 1
App. of EP	$apply(ep, t, b') \in S_{\mathcal{D}}^{\mathbf{P}}$	$\langle ep, t\rangle \in \boldsymbol{\epsilon}(\mathfrak{h})$	Lemma 4
Sensing	$sRes(l, n, b, b') \in S_{\mathcal{D}}^{\mathbf{P}}$	$\langle l, n\rangle \in sense(\boldsymbol{A}_{n,b}, \mathfrak{h}(n, b, S_{\mathcal{D}}^{\mathbf{P}}))$	Lemma 7
Action occ.	$occ(a, n, b) \in S_{\mathcal{D}}^{\mathbf{P}}$	$a \in \boldsymbol{A}_{n,b}$	Definition 2
Inference mechanisms			
Forward inertia	$kFwdInertia(l, t, n, b') \in S_{\mathcal{D}}^{\mathbf{P}}$	$\langle l, t\rangle \in add_{fwd}(\mathfrak{h})$	
Backward inertia	$kBackInertia(l, t, n, b') \in S_{\mathcal{D}}^{\mathbf{P}}$	$\langle l, t\rangle \in add_{back}(\mathfrak{h})$	
Causation	$kCause(l, t, n, b') \in S_{\mathcal{D}}^{\mathbf{P}}$	$\langle l, t\rangle \in add_{cause}(\mathfrak{h})$	
Positive postdiction	$kPosPost(l, t, n, b') \in S_{\mathcal{D}}^{\mathbf{P}}$	$\langle l, t\rangle \in add_{pd^{pos}}(\mathfrak{h})$	
Negative postdiction	$kNegPost(l, t, n, b') \in S_{\mathcal{D}}^{\mathbf{P}}$	$\langle l, t\rangle \in add_{pd^{neg}}(\mathfrak{h})$	
	where b' such that $hasChild(n, b, b', S_{\mathcal{D}}^{\mathbf{P}})$ and $t \leq n$.	where $\mathfrak{h} \in \Psi(\boldsymbol{A}_{n,b}, \mathfrak{h}(n, b, S_{\mathcal{D}}^{\mathbf{P}})$ and $t \leq n$.	
Auxiliary predicates			
Effect proposition	$hasEP(a, ep) \in S_{\mathcal{D}}^{\mathbf{P}}$	$ep \in \mathcal{EP}^a$	Lemma 9
Effect literal	$hasEff(ep, f) \in S_{\mathcal{D}}^{\mathbf{P}}$	$e(ep) = f$	Lemma 9
Condition literal	$hasCond(ep, f) \in S_{\mathcal{D}}^{\mathbf{P}}$	$f \in c(ep)$	Lemma 9
Knowledge proposition	$hasKP(a, f) \in S_{\mathcal{D}}^{\mathbf{P}}$	**determines**$(a, f) \in \mathcal{D}$	Lemma 9

Table 1: Relation between ASP formalization of $\mathcal{HPX}$ and its operational semantics

These implications emerge from the $\mathcal{HPX}$-logic program rules due to the stable model semantics: an atom can only exist in a stable model if it is in the head of an LP rule in which the body is compatible with the stable model. Implications (12a), (12d), (12h) and (12i) produce knowledge about pairs $\langle l, t\rangle$ independently from knowledge about other pairs $\langle l', t'\rangle$ for fixed n, b'. That is, to produce $knows(l, t, n+1, b')$ these implications do not directly depend on an atom $knows(l', t', n+1, b')$ in their body. This independence allows us to perform base steps for these rules. The base steps are derived with simple substitutions. For the rules that concern forward inertia (12b), backward inertia (12c), causation (12e), positive postdiction (12f) and negative postdiction (12g) we perform induction steps. The induction is complete because we consider *all* LP rules which can eventually generate a $knows/4$-atom. That is, all possible $knows(l, t, n+1, b')$ atoms are reached for arbitrary n, b. □

The Lemmata which we mention in Table 1 are defined and proven in Appendix B. The results concern only *soundness* of the ASP implementation wrt. the operational semantics. *Completeness* results are not provided because the ASP implementation is incomplete wrt. the operational semantics. The reason for incompleteness is that we have not found a solution to model the quantifications in the inertia and positive postdiction mechanisms of the operational semantics in terms of ASP. We address the incompleteness issue by demonstrating that the solvable fragment of the ASP implementation is useful. To this end, we conduct an empirical evaluation (Section 4.1), and we provide an extensive case study (Section 4.2), both based on the ASP implementation.

4. Empirical Evaluation, and a Case Study with a Robotic Wheelchair in a Smart Home

By approximating the $\mathcal{PWS}$-based approach of epistemic action theory we trade completeness for better computational complexity properties, and hence better computational performance. To this end, we are interested in how *good* this trade-off really is, i.e. how much performance is gained in relation to the number of problem instances that become unsolvable by the approximation. We show empirically that indeed merely around 15% of the problem instances within the investigated parameter space become unsolvable,

while the remaining 85% are solvable much more efficiently, i.e. in polynomial time.[7] Note that we consider projection problems in our evaluation, because they are much more fundamental then planning problems. The solvable fragment of planning problems depends on the solvable fragment of projection problems, and therefore, evaluating the projection problem instead of the planning problem, gives a much clearer view on the usefulness of our approach.
In addition to the empirical evaluation we demonstrate the reasonability of the solvable fragment by presenting a case study where $\mathcal{HPX}$ is used for action planning, abnormality detection and postdictive explanation in a real robotic smart home, namely the Bremen Ambient Assisted Living Lab (BAALL) [16]. We emphasize that postdiction is a crucial requirement to diagnose abnormalities in the scenario. Other approximations, e.g. the 0-approximation or the ω-approximation [26], are – in their basic form – not capable of solving the required reasoning tasks, because they do not support postdiction.[8]

4.1. *Trading Completeness for Efficiency: Empirical Evaluation*

In order to evaluate the solvable fragment of projection problems empirically, we count the number of pairs $\langle l, n\rangle$ (denoted $n_{\mathcal{HPX}}$) that occur in the leaf state of a transition tree that is generated by $\mathcal{HPX}$, given a domain $\mathcal{D}$ and a plan p of depth n. We compare this number with the number of fluent literals (denoted $n_{\mathcal{PWS}}$) that are known to hold after applying the same plan in the same domain under a $\mathcal{PWS}$-based semantics. To realize this experiment, we employ a naive implementation of the $\mathcal{A}_k$ semantics without static causal laws as answer set programming. The size of the solvable fragment with respect to a $\mathcal{PWS}$-based approach is quantified by computing the knowledge quotient $k_{hpx} = \frac{n_{\mathcal{PWS}}}{n_{\mathcal{HPX}}}$ for variable plan depth and a variable number of fluents.
The result is illustrated in the 3d-graph in Figure 1, with the number of actions on the x-axis, the number of fluents on the y-axis, and the knowledge quotient k_{hpx} on the z-axis.

4.1.1. *Problem instances*

For our experiments, we implemented a random problem generator to produce domains and plans within a finite problem space that is spanned by the following parameters:

- Number of fluents
- Number of actions in the narrative
- Fraction of sensing actions in the narrative
- Max. number of conditions per effect proposition

A problem with evaluations based on random problem generation is that the results might not be very realistic and reasonable. Therefore we respect a *reasonability heuristic* for problem instances that is based on heuristics known from action planning. This involves (among other aspects detailed below) assuring that a fluent is not sensed if it is already known, and an action with an effect that is already known to hold is not executed. For example, a reasonable agent would not open a door if it knows that the door is already open, nor would it sense the open-state of the door if it already knows whether the door is open or closed. We also assure that all problem instances are consistent in the sense of [8, Definition 6]. Overall, we make the following restrictions:

1. Fluents that are already known to hold are not sensed.
2. An action is not executed if all of its effects are already known to hold.
3. Actions do not have contradictory effects.
4. Actions do not have contradictory conditions.
5. An action is not executed if at least one of its conditions is known not to hold.
6. An action is not executed if all its conditions are known to hold and the knowledge state that would result from executing the action has already been reached before.[9]

All applied restrictions resemble common heuristics used in action planning. We provide an example of a randomly generated problem instance in Appendix A.2.

[7] This percentage holds for cases with 11 fluents and sequences of 28 actions, where 25% of all actions are sensing actions. Similar results are achieved for other parametrizations; consider Section 4.1 for details.

[8] In principle, the extended 0-approximation semantics by [27] allows one to work around this problem by using static causal laws. However, as we demonstrate in [8, Example 4], this approach is not elaboration tolerant.

[9] This is necessary to avoid cyclic state transitions, such as a door that is is opened and closed over and over again. The restriction that at least one condition of an action that leads to an already reached knowledge state must be unknown assures, that the action has at least the potential to trigger additional knowledge gain about the condition by means of postdiction.

4.1.2. Evaluation

Recall that our evaluation of the solvable fragment is based on the number of known fluents in a leaf state of the transition tree that emerges from a domain $\mathcal{D}$ and a plan p. In order to compare the number of known fluents generated by $\mathcal{HPX}$ with the number of known fluents generated by our naive $\mathcal{A}_k$ implementation, it not necessary to compare all leaf states in the transition tree; it is sufficient to compare only one leaf state and to compute the quotient k_{hpx} for this state. This simplifies the reasoning process in that we only have to consider one valid model in the $\mathcal{A}_k$ semantics.

In terms of the ASP implementation of $\mathcal{HPX}$, the simplification means that we have to disable the branching mechanism. That is, for a sensing action described through a knowledge proposition **determines**(a_s, f^s), we model the occurrence of a_s at a step n in a branch b not only by adding the atom $occ(a_s, n, b)$ to the logic program, but also by adding an atom $knows(f^s, n, n, b)$ or $knows(\neg f^s, n, n, b)$. According to (F5f) and (F5g), this prevents the generation of $sRes/4$ atoms and hence the generation of branches. This behavior is therefore equivalent to considering only one *rational model* (see [26, Def. 4])[10] in the $\mathcal{A}_k$ semantics.

To obtain significant data, we run 40 random problem instances per parametrization and compute the average computation time and number of known fluents in the stable models of both ASP implementations, i.e. the ASP formalization of $\mathcal{HPX}$ and our naive $\mathcal{A}_k$ implementation.The computations were performed using the ASP solver *clingo* v. 3.05 [11], on a Intel i7 machine at 2400 Mhz with 4GB RAM.

4.1.3. Results

We are interested in the size of the solvable fragment generated by $\mathcal{HPX}$ in relation to the $\mathcal{PWS}$-based $\mathcal{A}_k$ semantics and in performance gain. To this end, we ran the random problem generator for different parametrizations and computed the average computation time and fraction of known fluents over 30 runs for each parametrization. Figure 1 represents the fraction of knowledge generated by $\mathcal{HPX}$ wrt. $\mathcal{A}_k$ for a narrative with up to 30 actions and a domain with up to 12 fluents. 25% of actions (rounded down) in the narratives were sensing actions, and all effect propositions in the domain had up to three condition literals. The order of sensing and non-sensing actions was random. Since the more actions that occur, the more fluent values are known after action execution, there exist no reasonable narratives for cases where the number of actions is high wrt. the number of fluents. In consequence, the graph area does not display data for such cases.

► **Solvable fragment.** For most reasonable narratives, the solvable fragment is above 85%, in particular for domains with many fluents and few actions. The solvable fragment of our approach is only significantly smaller in extreme cases, where the number of actions is ver high compared to the number of fluents. As an example, consider the case where 11 actions are executed in a domain with only 3 fluents. Here our approach generates knowledge for 40% of all fluents in average, compared to $\mathcal{A}_k$. Even though our reasonability heuristic is still respected in such cases, we think that respective problems are still quite uncommon in reality; in particular for planning purposes where the number of actions is usually kept as small as possible.

We should also mention that we tried other values for the fraction of sensing actions and the number of condition literals, but we could not observe significant effects of these parameters on the average size of the solvable fragment.

► **Computational performance.** In addition to the size of the solvable fragment of $\mathcal{HPX}$ we are interested in the gain of computational performance that results from the better complexity properties compared to a naive $\mathcal{PWS}$-based approach. This is illustrated in Figure 2. The z-axis represents how many times $\mathcal{HPX}$ is faster than the $\mathcal{PWS}$-based naive implementation of $\mathcal{A}_k$ depending on the number of actions and the number of fluents per problem instance. The logarithmic scale is required to capture the exponential nature of $\mathcal{A}_k$. Note that for problems with less than four actions and four fluents no meaningful comparison of performance is possible, because such problems were solved in less than one millisecond. We use the same parametrization as for the experiment to determine the solvable fragment, i.e. 25% sensing actions and up to three condition literals per effect proposition. We also tried alternative parametrizations, and found that these produce similar results.

The measured computation time involves ASP preprocessing, grounding and solving. It turns out that grounding takes most time. In particular for $\mathcal{A}_k$, where grounding generates the exponential number of possible worlds – a step that is not required in $\mathcal{HPX}$. For example, for an $\mathcal{A}_k$ problem instance with 12 fluents and 25 actions, clingo took 56.78 sec. in total, where 18.46 sec. were required for preprocessing, 38.31 sec. for grounding and 0.01 sec. for the actual solving. The same problem instance for $\mathcal{HPX}$ was solved 25 times faster. It took 0.73 sec. in total, where 0.67 sec. was preprocessing, 0.05 sec. grounding and 0.01 sec. solving time.

4.2. Case Study: Abnormality Detection in a Smart Home

To demonstrate the general usefulness of postdictive reasoning we present a use case as depicted in Figure 3. The illustrated scenario is fully implemented for the Bremen Ambient Assisted Living Lab (BAALL), i.e. the ASP solver directly controls doors

[10]Considering only one model intuitively means to consider one possible world in which an agent with incomplete knowledge acts. The agent's incomplete knowledge is modelled with multiple possible worlds that are compatible with the actually considered possible world. For example, if it is unknown whether a door is open or closed, then this results in two possible worlds. We select one of these worlds (one rational model) and model an agent's incomplete knowledge about this world model again by multiple possible worlds. For details consider [26].

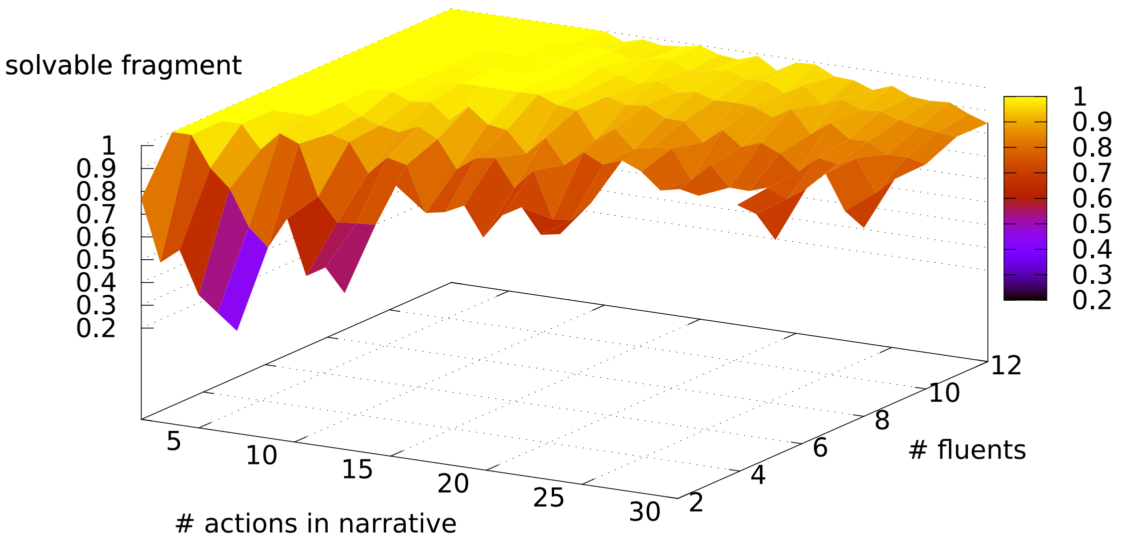


Figure 1: Solvable fragment of $\mathcal{HPX}$ wrt. $\mathcal{A}_k$.

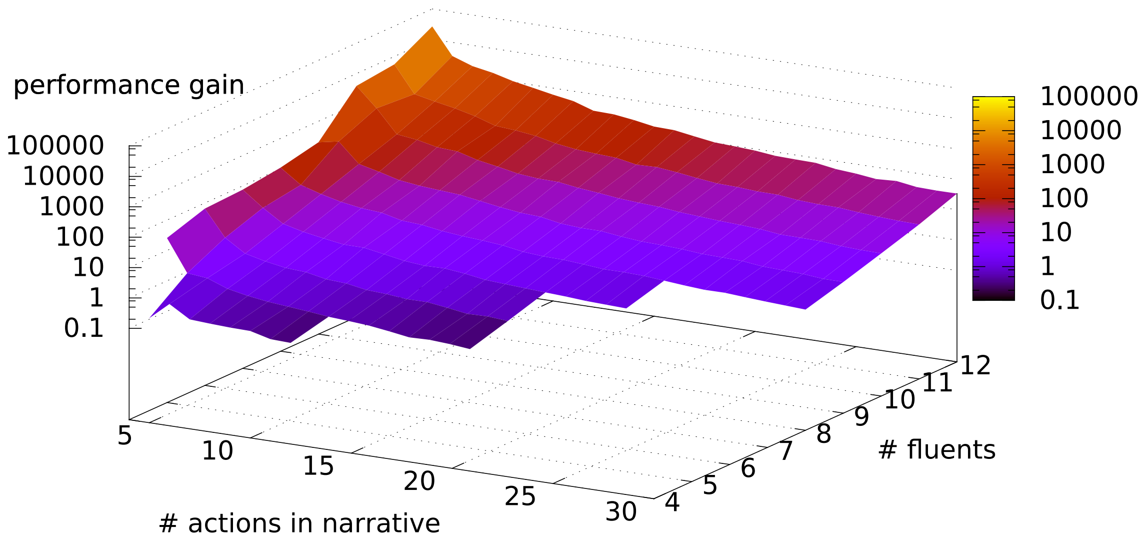


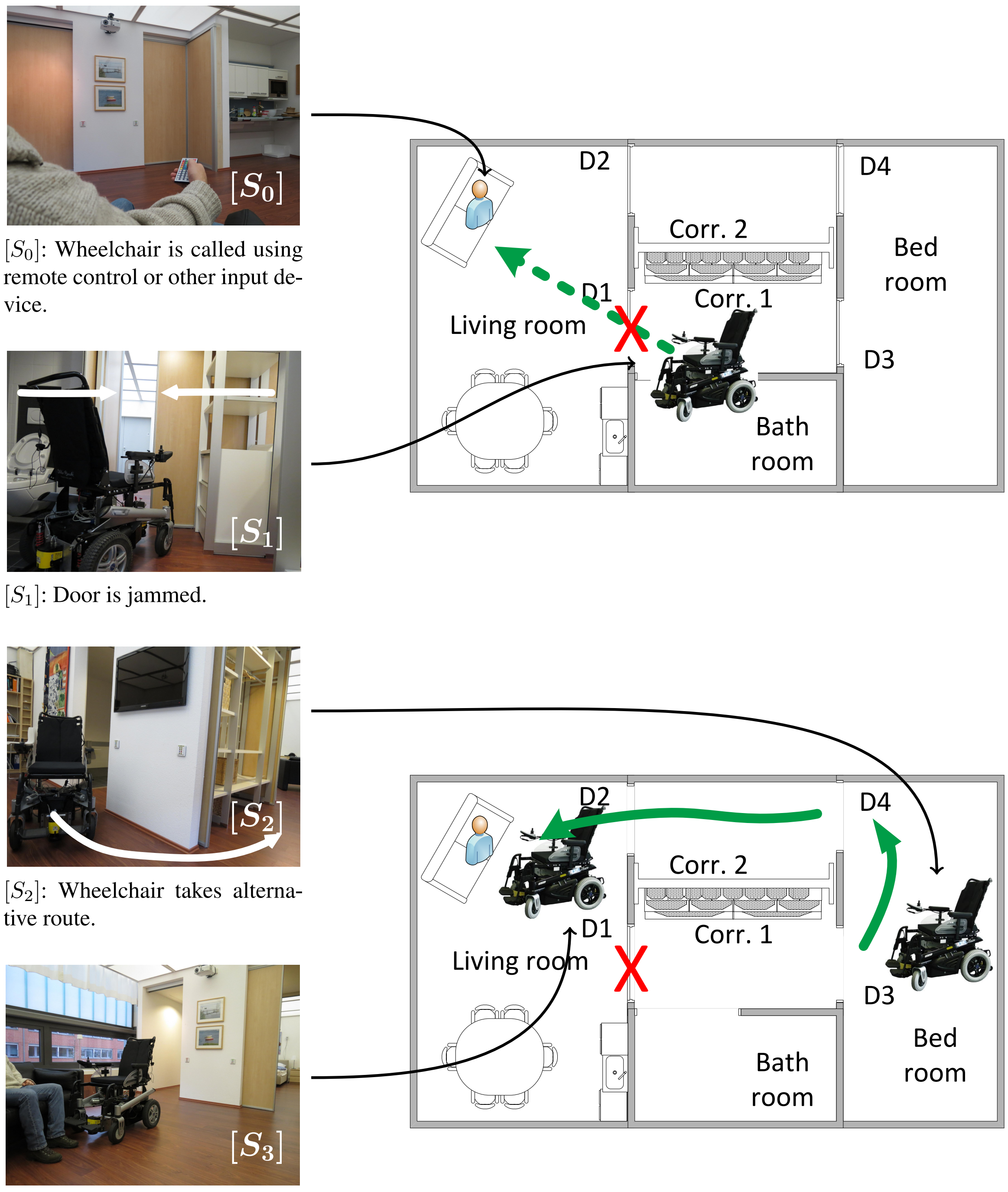
Figure 2: Performance gain in relation to number of actions and number of fluents per domain

and the wheelchair in the BAALL.[11] The BAALL has (automatic) sliding doors, and sometimes an obstacle (e.g. a box or a chair) accidentally blocks the door such that it opens only half way. In this case, the planning component in the overall system should be able to postdict such an abnormality and to find an alternative route for robotic vehicles which would usually pass through the door. A simplified domain description is as follows:

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

An action `open_door` causes a door to be open if it is not jammed. The action `drive` has the effect that the robot is in the living room

[11]A video demonstration providing a general overview of the kinds robot control scenarios that we address with the BAALL Lab / wheelchair robot can be accessed at http://www.manfred.eppe.eu/the-hpx-online-planning-system/ (please note that the exact scenario in the video corresponds to a minor variation of the domain theory presented in the case-study).



[S_0]: Wheelchair is called using remote control or other input device.

[S_1]: Door is jammed.

[S_2]: Wheelchair takes alternative route.

[S_3]: Destination reached.

Figure 3: Case study: abnormality detection in the smart home *BAALL*

(which is behind the door) if the door is open. `sense_open` can be executed to determine the open-state of the door. Initially the door is closed, and the goal of a corresponding planning task is that the wheelchair must arrive in the living room. Consequently, we query for a plan where the fluent `in_liv` must be true in at least one leaf state of the transition tree. For simplicity, we have no executability conditions in this case.

Trace: Conditional Planning with Abnormalitiy Postdiction

Consider the situation where a person instructs a command to reach a location, e.g. the sofa, to the wheelchair [S_0]. An optimal plan to achieve this goal is to pass D1. However, if D1 does not open because it is jammed, then a more error tolerant plan is required: [S_1] open D1 and verify if the action succeeded by sensing the door status. If the door is open, drive through the door and

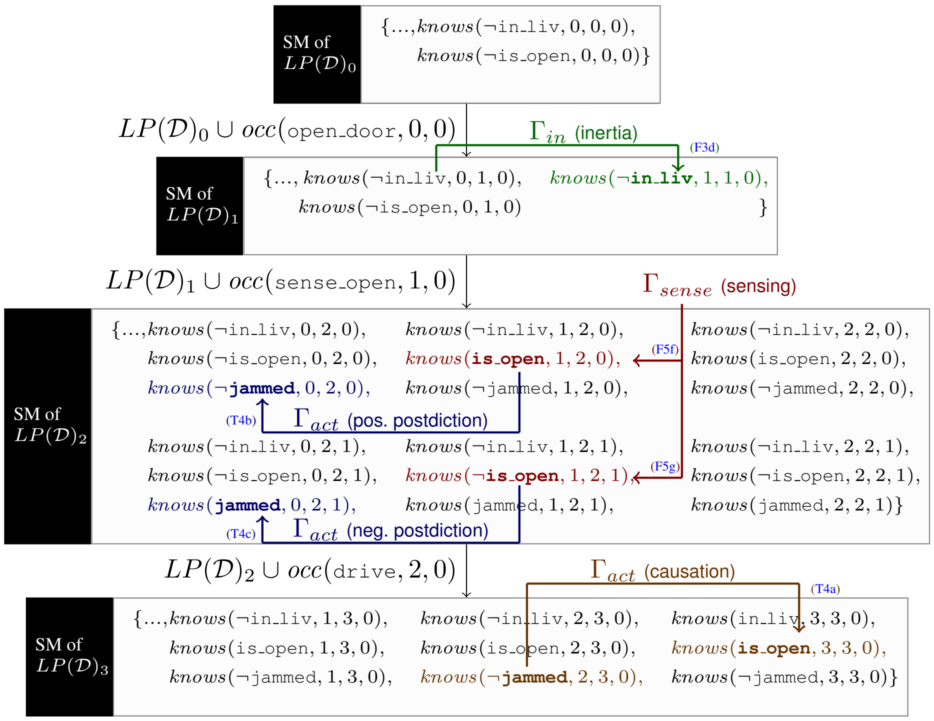

SM of $LP(\mathcal{D})_0$
$\{\ldots, knows(\neg \texttt{in\_liv}, 0, 0, 0),$ $knows(\neg \texttt{is\_open}, 0, 0, 0)\}$
$LP(\mathcal{D})_0 \cup occ(\texttt{open\_door}, 0, 0)$
Γ_{in} (inertia)
(F3d)
SM of $LP(\mathcal{D})_1$
$\{\ldots, knows(\neg \texttt{in\_liv}, 0, 1, 0),$ $knows(\neg \texttt{in\_liv}, 1, 1, 0),$ $knows(\neg \texttt{is\_open}, 0, 1, 0)$ $\}$
$LP(\mathcal{D})_1 \cup occ(\texttt{sense\_open}, 1, 0)$
Γ_{sense} (sensing)
SM of $LP(\mathcal{D})_2$
$\{\ldots, knows(\neg \texttt{in\_liv}, 0, 2, 0),$ $knows(\neg \texttt{in\_liv}, 1, 2, 0),$ $knows(\neg \texttt{in\_liv}, 2, 2, 0),$
$knows(\neg \texttt{is\_open}, 0, 2, 0),$ $knows(\texttt{is\_open}, 1, 2, 0),$ (F5f) $knows(\texttt{is\_open}, 2, 2, 0),$
$knows(\neg \texttt{jammed}, 0, 2, 0),$ $knows(\neg \texttt{jammed}, 1, 2, 0),$ $knows(\neg \texttt{jammed}, 2, 2, 0),$
(T4b) Γ_{act} (pos. postdiction)
$knows(\neg \texttt{in\_liv}, 0, 2, 1),$ $knows(\neg \texttt{in\_liv}, 1, 2, 1),$ $knows(\neg \texttt{in\_liv}, 2, 2, 1),$
$knows(\neg \texttt{is\_open}, 0, 2, 1),$ $knows(\neg \texttt{is\_open}, 1, 2, 1),$ (F5g) $knows(\neg \texttt{is\_open}, 2, 2, 1),$
$knows(\texttt{jammed}, 0, 2, 1),$ $knows(\texttt{jammed}, 1, 2, 1),$ $knows(\texttt{jammed}, 2, 2, 1)\}$
(T4c) Γ_{act} (neg. postdiction)
$LP(\mathcal{D})_2 \cup occ(\texttt{drive}, 2, 0)$
Γ_{act} (causation)
(T4a)
SM of $LP(\mathcal{D})_3$
$\{\ldots, knows(\neg \texttt{in\_liv}, 1, 3, 0),$ $knows(\neg \texttt{in\_liv}, 2, 3, 0),$ $knows(\texttt{in\_liv}, 3, 3, 0),$
$knows(\texttt{is\_open}, 1, 3, 0),$ $knows(\texttt{is\_open}, 2, 3, 0),$ $knows(\texttt{is\_open}, 3, 3, 0),$
$knows(\neg \texttt{jammed}, 1, 3, 0),$ $knows(\neg \texttt{jammed}, 2, 3, 0),$ $knows(\neg \texttt{jammed}, 3, 3, 0)\}$

Figure 4: Abnormality detection as postdiction with *h-approximation*

approach the user. Else there is an abnormality: in this case open and pass D3 [S_2]; drive through the bedroom; pass D4 and D2; and finally approach the sofa [S_3].[12] A transition tree is provided in Figure 4.

Initially, wheelchair *Rolland* is outside the living room ($\neg \texttt{in\_liv}$) and the weak goal is that the robot is inside the living room. The robot can open the door ($\texttt{open\_door}$) to the living room. Unfortunately, since the door may be jammed, opening the door does not always work, i.e. there may be an abnormality. However, the robot can perform sensing to verify whether the door is open ($\texttt{sense\_open}$) and then postdict whether or not there is an abnormality in opening the door.

This mechanism is illustrated in Figure 4. Initially (at step $n = 0$ and branch $b = 0$) it is known that the robot is in the corridor at step $t = 0$ (denoted by $knows(\neg \texttt{in\_liv}, 0, 0)$). The first action is opening the door, i.e. the stable model contains the atom $occ(\texttt{open\_door}, 0, 0)$. Inertia holds for $\neg \texttt{in\_liv}$, because nothing happened that could have initiated $\texttt{in\_liv}$. Consequently, rules (F3a) – (F3c) trigger $kNotInit(\texttt{in\_liv}, 0, 0, 0)$ and (F3f) triggers $knows(\neg \texttt{in\_liv}, 0, 1, 0)$. In turn, the forward inertia rule (F3d) causes atom $knows(\neg \texttt{in\_liv}, 1, 1, 0)$ to hold. Next, sensing happens, i.e. $occ(\texttt{sense\_open}, 1, 0)$. According to rule (F5f), the positive result is assigned to the original branch and $sRes(\texttt{is\_open}, 1, 0, 0)$ is produced. With rule (F5g), the negative sensing result is assigned to some child branch $b' = 1$. In the example we have $sRes(\neg \texttt{is\_open}, 1, 0, 1)$, such that together with (F5k) $knows(\neg \texttt{is\_open}, 1, 2, 1)$ is produced. This result triggers the negative postdiction rule (T4c) and knowledge about an abnormality concerning the opening of the door is produced: $knows(\texttt{jammed}, 0, 2, 1)$. Consequently, the wheelchair has to follow another route to achieve the goal.

For branch 0, we have $knows(\texttt{is\_open}, 1, 2, 0)$ after the sensing. This result triggers the positive postdiction rule (T4b): because $knows(\neg \texttt{is\_open}, 0, 2, 0)$ and $knows(\texttt{is\_open}, 1, 2, 0)$ hold, one can postdict that there was no abnormality when $\texttt{open\_door}$ occurred: $knows(\neg \texttt{jammed}, 0, 2, 0)$. Finally, the robot can drive through the door: $occ(\texttt{drive}, 2, 0)$ and the causation rule (T4a) triggers knowledge that the robot is in the living room at step 3: $knows(\texttt{in\_liv}, 3, 3, 0)$.

5. Conclusion and Outlook

We demonstrate how the *h-approximation* ($\mathcal{HPX}$), an epistemic action theory based on knowledge histories that is presented in [8], can be implemented in the declarative non-monotonic framework of answer set programming (ASP) [2]. $\mathcal{HPX}$ has a lower computational complexity than action theories based on a possible worlds semantics, which comes at the cost that the theory is not complete. The implementation presented in this work contributes to the state of the art by allowing us to perform an empirical evaluation concerning the solvable solvable fragment of the $\mathcal{HPX}$ approach. The result of this evaluation is, that approximately 85% of knowledge generated with an exponential approach based on possible worlds can also be generated with $\mathcal{HPX}$ for most of

[12] Abnormalities are considered on the alternative route as well but skipped here for brevity.

the parameterizations described in Sec. 4.1. We consider this to be a good tradeoff when looking at the theoretical and empirical complexity results, which show that in the polynomial complexity hierarchy, $\mathcal{HPX}$ is one level below comparable $\mathcal{PWS}$-based approaches like the language $\mathcal{A}_k$[26] (NP in $\mathcal{HPX}$ vs. Σ_2^P in $\mathcal{A}_k$ for the planning problem). In other words, by using $\mathcal{HPX}$ and going down a whole complexity class in the polynomial hierarchy, one can still find 85% of solutions for most planning problem instances.

As future work, it would be interesting to investigate this relation with other epistemic action theories like the 0-approximation of knowledge [26] or the EFEC formalism [19]. In particular EFEC is an interesting candidate, since it also considers the temporal knowledge dimension that we advocate.

In addition to the empirical evaluation concernig the solvable fragment, we also demonstrate the general usefulness of $\mathcal{HPX}$ and its postdictive reasoning capabilities for diagnosis tasks. Towards this, we present a case study to demonstrate how abnormalities are diagnosed and accounted for within a real-time robot control task in a smart home (see Section 4.2).

The ASP implementation of the $\mathcal{HPX}$ reasoning system provides a robust, declarative framework to further extend and apply our reasoning system from the viewpoint of incorporating new features (e.g., functional fluents using ASPMT [3]), and reaching out to other research communities where reasoning about action and change using KR-based methods could be valuable. Indeed, computing explanations and establishing hypotheses based on observations and partial knowledge is an important capability essential in a wide range of applications. Robotics has been chosen as a prime area of demonstration in this paper; however, it is by no means the only one, or the most important. As described in [8], we envision other areas and domains of applications where a general capability to compute explanations and establish hypotheses exist. As an example consider (i) decision support systems in the field of medicine, (ii) dynamic geospatial systems where changes in qualitative spatio-temporal relationships between complex aggregate entities at the geospatial scale need to be analysed using high-level commonsense reasoning [4], (iii) spatial design cognition and computation, which has the need to perform diagnosis and compute counterfactual scenarios for incremental or iterative design refinementand, or, (iv) assistive technologies in crime, forensics, and legal reasoning, which often involve perspective-dependent narratives of temporally grounded beliefs, and the formation of new beliefs or invalidation of old beliefs based on new knowledge.

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" [9]
- "Narrative based postditive reasoning for Cognitive Robotics" [7]

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.

References

[1] Babb, J., & Lee, J. (2013). Cplus2ASP : Computing Action Language C + in Answer Set Programming. In *International Conference on Logic Programming and Nonmonotonic Reasoning*.

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

[3] Bartholomew, M., & Lee, J. (2013). Functional stable model semantics and answer set programming modulo theories. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

[4] 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).

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

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

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

[8] Eppe, M., & Bhatt, M. (2015). A history based approximate epistemic action theory for efficient postdictive reasoning. *Journal of Applied Logic*, *this issue*.

[9] 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*.

[10] Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2012). *Answer Set Solving in Practice*. Morgan and Claypool.

[11] Gebser, M., Kaminski, R., König, A., & Schaub, T. (2011). Advances in gringo series 3. In *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning* X.

[12] Gelfond, M., & Lifschitz, V. (1988). The Stable Model Semantics for Logic Programming. In *Proceedings of the International Conference on Logic Programming (ICLP)*.

[13] Gelfond, M., & Lifschitz, V. (1993). Representing action and change by logic programs. *The Journal of Logic Programming*, *17*, 301–321.

[14] Gelfond, M., & Lifschitz, V. (1998). Action languages. *Electronic Transactions on Artificial Intelligence*, *3*, 1–23.

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

[16] Krieg-Brückner, B., Röfer, T., Shi, H., & Gersdorf, B. (2010). Mobility Assistance in the Bremen Ambient Assisted Living Lab. *GeroPsych: The Journal of Gerontopsychology and Geriatric Psychiatry*, *23*, 121–130.

[17] Lee, J., & Palla, R. (2010). Situation calculus as answer set programming. *AAAI Proceedings*, .

[18] Lee, J., & Palla, R. (2012). Reformulating the situation calculus and the event calculus in the general theory of stable models and in answer set programming. *Journal of Artificial Intelligence Research*, *43*, 571–620.

[19] 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*.

[20] McCarthy, J., & Hayes, P. (1969). Some philosophical problems from the standpoint of artificial intelligence. *Machine intelligence*, *4*, 463–502.

[21] 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.

[22] Mueller, E. (2004). Event Calculus Reasoning Through Satisfiability. *Journal of Logic and Computation*, *14*.

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

[24] Reiter, R. (1991). The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz (Ed.), *Artificial intelligence and mathematical theory of computation* (pp. 359–380). Academic Press Professional, Inc.

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

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

[27] 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.

Appendix A. Source Code

Appendix A.1. Foundational Theory of the Offline ASP Formalization of $\mathcal{HPX}$

The foundational part of the ASP formalization is provided in Listing 1. Note that for brevity we use a predicate `l/1` instead of `literal/1` to denote literal declarations, and similarly `f/1` instead of `fluent/1`.

Listing 1: Foundational theory (Γ_{hapx}) of the ASP implementation of $\mathcal{HPX}$

```
▶ F1. Auxiliaries (Γaux)
s(0..maxS).
br(0..maxBr).
neq(B,B1) :- B != B1, br(B), br(B1).
l(neg(F)):- f(F).
l(F):- f(F).
complement(neg(F),F) :- f(F).
complement(L1,L2) :- complement(L2,L1).

▶ F2. Concurrency (Γconc)
apply(EP,N,B) :- hasEP(A,EP), occ(A,N,B).
:- apply(EP1,T,B), hasEff(EP1,L), apply(EP2,T,B), hasEff(EP2,L), EP1 != EP2, br(B), l(L).

▶ F3. Inertia (Γin)
kNotSet(L,T,N,B) :- not kMaySet(L,T,B), uBr(N,B), s(T),l(L).
kMaySet(L,T,B) :- apply(EP,T,B), hasEff(EP,L).
kNotSet(L,T,N,B) :- apply(EP,T,B), hasEff(EP,L), hasCond(EP,L1), knows(L2,T,N,B), complement(L1,L2),s(T).
knows(L,T,N,B) :- knows(L,T-1,N,B), kNotSet(L1,T-1,N,B),        complement(L,L1), T<=N.
knows(L,T,N,B) :- knows(L,T+1,N,B), kNotSet(L,T,N,B), N > T.
knows(L,T,N,B) :- knows(L,T,N-1,B), uBr(N,B), N <= maxS.

▶ F5. Sensing and Branching (Γsense)
uBr(0,0).
sNextBr(N,B) :- sRes(L,N,B,B1).
uBr(N,B) :- uBr(N-1,B), not sNextBr(N-1,B), s(N).
kw(F,T,N,B) :- knows(F,T,N,B).
kw(F,T,N,B) :- knows(neg(F),T,N,B).
sRes(F,N,B,B):-occ(A,N,B),hasKP(A,F),not kw(neg(F),N,N,B),s(N).
1{sRes(neg(F),N,B,B1) : neq(B,B1)}1 :- occ(A,N,B), hasKP(A,F),    not kw(F,N,N,B), s(N).
:- sRes(L,N,B,B1), uBr(N,B1), l(L), neq(B,B1).
:- 2{sRes(L,N,B,B1) :br(B) : l(L)},br(B1),s(N).
uBr(N,B1) :- sRes(L,N-1,B,B1), s(N).
knows(L,N-1,N,B1) :- sRes(L,N-1,B,B1), s(N).
:- 2{occ(A,N,B) : hasKP(A,_)}, br(B),s(N).
knows(L,T,N,B1) :- sRes(_,N-1,B,B1), knows(L,T,N-1,B), N>=T.
apply(EP,T,B1) :- sRes(_,N,B,B1), apply(EP,T,B), N>=T.

▶ F4. inference mechanisms (Γinfer)
knows(L,T,N,B) :- kCause(L,T,N,B), uBr(N,B).
knows(L,T,N,B) :- kPosPost(L,T,N,B), uBr(N,B).
knows(L,T,N,B) :- kNegPost(L,T,N,B), uBr(N,B).

▶ F6. Plan verification (Γverify)
notWG(N,B) :- wGoal(L), uBr(N,B), not knows(L,N,N,B), l(L).
allWGAchieved(N) :- not notWG(N,B), uBr(N,B).
:- not allWGAchieved(maxS).
notSG(N,B) :- sGoal(L), uBr(N,B), not knows(L,N,N,B), l(L).
:- notSG(maxS,B), uBr(maxS,B).
notGoal(N,B) :- notSG(N,B).
notGoal(N,B) :- notWG(N,B).

▶ F7. Plan generation and optimization (Γplan)
% Sequential Planning:
1{occ(A,N,B): a(A)}1 :- uBr(N,B), notGoal(N,B), N < maxS.
% Concurrent Planning:
% 1{occ(A,N,B) : a(A)} :- uBr(N,B), notGoal(N,B), N < maxS.
#minimize {occ(_,_,_) @ 1}.
```

Appendix A.2. Randomly Generated Problem Instance for Empirical Evaluation

The following is a randomly generated problem instance as used in the Empirical Evaluation in Section 4.1. This example involves ten fluents, twelve non-sensing actions and three sensing actions.

Listing 2: Random problem instance

```
%%%%%%%%%%%      Automatically generated random narrative      %%%%%%%%%%%
% Parameters:
% 10 fluents
% 12 non-sensing actions
% 3 sensing actions
%%%%%%%%%%%    Fluents      %%%%%%%%%%%%
f(1).
f(2).
f(3).
f(4).
f(5).
f(6).
f(7).
f(8).
f(9).
f(10).
%%%%%%%%%%%    Non-sensing actions     %%%%%%%%%%%%
hasEff(act_0,5).
hasCond(act_0,neg(5)).
hasCond(act_0,neg(1)).
hasCond(act_0,neg(7)).
hasEff(act_1,neg(6)).
hasCond(act_1,neg(2)).
hasCond(act_1,9).
hasCond(act_1,neg(5)).
hasEff(act_2,4).
hasCond(act_2,9).
hasCond(act_2,2).
hasCond(act_2,6).
hasEff(act_4,9).
hasCond(act_4,8).
hasCond(act_4,1).
hasCond(act_4,2).
hasEff(act_6,3).
hasCond(act_6,10).
hasCond(act_6,neg(9)).
hasCond(act_6,6).
hasEff(act_7,10).
hasCond(act_7,6).
hasCond(act_7,4).
hasCond(act_7,8).
hasEff(act_8,neg(9)).
hasCond(act_8,neg(10)).
hasCond(act_8,4).
hasCond(act_8,6).
hasEff(act_9,neg(9)).
hasCond(act_9,1).
hasCond(act_9,neg(10)).
hasCond(act_9,neg(3)).
hasEff(act_10,neg(4)).
hasCond(act_10,6).
hasCond(act_10,neg(10)).
hasCond(act_10,neg(5)).
hasEff(act_11,neg(2)).
hasCond(act_11,3).
hasCond(act_11,6).
hasCond(act_11,2).
hasEff(act_13,neg(8)).
hasCond(act_13,2).
hasCond(act_13,8).
hasCond(act_13,1).
hasEff(act_14,neg(2)).
hasCond(act_14,6).
hasCond(act_14,2).
hasCond(act_14,neg(7)).
```

```
%%%%%%%%%%%    Narrative     %%%%%%%%%%%%
occ(act_0,0).
occ(act_1,1).
occ(act_2,2).
sense(1,3).
occ(act_4,4).
sense(neg(3),5).
occ(act_6,6).
occ(act_7,7).
occ(act_8,8).
occ(act_9,9).
occ(act_10,10).
occ(act_11,11).
sense(neg(10),12).
occ(act_13,13).
occ(act_14,14).
```

The following rules are added to the Logic Program to deal with the abbreviated atoms $sense/2$ and $occ/2$. They also avoid the branching as described in Section 4.1.2.

$$\begin{aligned} knows(L,N,N,B) &\leftarrow sense(L,N), uBr(N,B). \\ occ(sense(L),N) &\leftarrow sense(L,N). \\ hasKP(sense(L),L) &\leftarrow sense(L,_{)}\,. \\ occ(A,N,B) &\leftarrow occ(A,N), uBr(N,B). \end{aligned} \tag{A.1}$$

Appendix B. Soundness of ASP Implementation wrt. $\mathcal{HPX}$ Semantics

Recall the central soundness Theorem 1 from Section 3.6:

Theorem 2 (Soundness for knowledge atoms for single state transitions). *For all n, t, b, if there exists a branch b' such that $hasChild(n,b,b',S_{\mathcal{D}}^{\boldsymbol{P}})$ and $knows(l,t,n{+}1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}}$, then there exists an h-state $\mathfrak{h} \in \Psi(\boldsymbol{A}_{n,b}, \mathfrak{h}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}))$ such that $\mathfrak{h} \models \langle l,t\rangle$. Formally:*

$$\begin{aligned} &\forall n,b,b' : \big(hasChild(n,b,b',S_{\mathcal{D}}^{\boldsymbol{P}})\big) \Rightarrow \\ &\qquad \big(\exists \mathfrak{h} \in \Psi(\boldsymbol{A}_{n,b}, \mathfrak{h}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}})) : \forall l,t : (knows(l,t,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Rightarrow \mathfrak{h} \models \langle l,t\rangle\,)\big) \end{aligned}$$

As discussed in 3.6, to prove the Theorem, we first eliminate the $\exists$-quantification over $\mathfrak{h}$ with a case distinction, and then perform induction over the structure of implications (12) for pairs $\langle l,t\rangle$ (see Appendix B.1). This requires some auxiliary Lemmata (Appendix B.2) and relies on other inductive proofs, in particular concerning the application of effect propositions (Appendix B.3), sensing results (Appendix B.4) and auxiliary predicates (Appendix B.5).

Appendix B.1. Soundness of Knowledge Atoms

Proof of Theorem 1:
In order to prove (11), we first substitute the transition function according to [8, Eq. (6)]: $\Psi(\boldsymbol{A}_{n,b}, \mathfrak{h}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}})) = \bigcup_{k \in sense(\mathbf{A},\mathfrak{h}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}))} eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \cup k\rangle)$. It is further easy to see that $now\big(\mathfrak{h}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}})\big) = n$. Hence we obtain (B.1).

$$\begin{aligned} &\forall n,b,b' : \big(hasChild(n,b,b',S_{\mathcal{D}}^{\boldsymbol{P}})\big) \Rightarrow \\ &\qquad \Big(\exists \mathfrak{h} \in \bigcup_{k \in sense(\mathbf{A},\mathfrak{h}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}))} eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \cup k\rangle) : \forall l,t : (knows(l,t,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Rightarrow \mathfrak{h} \models \langle l,t\rangle\,)\Big) \\ &\quad \text{with } t \le n \text{ and } \boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \cup \{\langle a,n\rangle \,|a \in \boldsymbol{A}_{n,b}\} \end{aligned} \tag{B.1}$$

To prove that (B.1) holds, we perform induction over the structure of (12) for pairs $\langle l,t\rangle$. To this end, we first determine under which conditions $hasChild(n,b,b',S_{\mathcal{D}}^{\boldsymbol{P}})$ becomes true and we eliminate the $\exists$ quantification over h-states. Therefore we distinguish two cases as follows.

Case 1: $\exists l' : (sRes(l', n, b, b') \in S^{\boldsymbol{P}}_{\mathcal{D}})$ *(With Sensing Result)*
We prove that (B.1) holds if sensing results are obtained, i.e. if (B.2) holds.

$$\exists l' : (sRes(l', n, b, b') \in S^{\boldsymbol{P}}_{\mathcal{D}}) \tag{B.2}$$

The following formulae are universally quantified over those n, b, b' for which (B.2) holds. The case distinction allows us to simplify (B.1) and we make the following substitutions and generalizations for this case.

(B.1)

By case distinction (B.2) and definition of $hasChild$ (8) we have:

$$\exists l' : (sRes(l', n, b, b') \in S^{\boldsymbol{P}}_{\mathcal{D}}) \Rightarrow hasChild(n, b, b', S^{\boldsymbol{P}}_{\mathcal{D}})$$

Hence, in order to prove (B.1) it is sufficient to prove (B.3)

$$\exists \mathfrak{h} \in \bigcup_{k \in sense(\mathbf{A}, \mathfrak{h}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}))} eval(\boldsymbol{\alpha}', \boldsymbol{\kappa}(n, b, S^{\boldsymbol{P}}_{\mathcal{D}}) \cup k) : \forall l, t : \left(knows(l, t, n+1, b') \in S^{\boldsymbol{P}}_{\mathcal{D}}\right) \Rightarrow (\mathfrak{h} \models \langle l, t \rangle) \tag{B.3}$$

with $t \leq n$ and $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n, b, S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a, n\rangle \,|a \in \boldsymbol{A}_{n,b}\}$

(B.3)

We now eliminate the $\exists$-quantification by generalizing (B.3). The case distinction (B.2) states that there exists at least one literal l' for which $sRes(l', n, b, b') \in S^{\boldsymbol{P}}_{\mathcal{D}}$. In the following we consider an arbitrary literal l^s such that:

$$sRes(l^s, n, b, b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \tag{B.4}$$

That is, we presume that the following formulae are implicitly universally quantified over l^s for which (B.4) holds. Further, by Lemma 7:

$$\left(sRes(l^s, n, b, b') \in S^{\boldsymbol{P}}_{\mathcal{D}}\right) \Rightarrow \left(sense(\mathbf{A}, \mathfrak{h}(n, b, S^{\boldsymbol{P}}_{\mathcal{D}})) = \{\{\langle l^s, n\rangle\}, \{\langle \overline{l^s}, n\rangle\}\}\right)$$

Hence, in order to show that (B.3) holds, it is sufficient to show that (B.5) holds.

$$\forall l, t : \left(\left(knows(l, t, n+1, b') \in S^{\boldsymbol{P}}_{\mathcal{D}}\right) \Rightarrow \left(eval(\boldsymbol{\alpha}', \boldsymbol{\kappa}(n, b, S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l^s, n\rangle) \models \langle l, t\rangle\right)\right) \tag{B.5}$$

with $t \leq n$ and $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n, b, S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a, n\rangle \,|a \in \boldsymbol{A}_{n,b}\}$

In the following we prove (B.5) by induction over the structure of implications (12) which generate $knows/4$ atoms.

Base Steps for Case 1

1. *Initial Knowledge:* $\{\langle l, t\rangle \,|knows(l, t, n+1, b')$ is produced by (T2)$\}$
   Implication (12a) generates knowledge for step 0 only, i.e. according to (12a) it must hold that if an atom $knows(l, t, n+1, b')$ is generated then $n + 1 = 0$. However, since by Definition 2 we consider $n \geq 0$, (T2) can not produce an atom $knows(l, t, n+1, b)$. Therefore this case does not apply.[13]

[13]Note that this soundness proof concerns state transitions. Since the initial knowledge to which this base step refers is not the result of a state transition, it does not have to be considered.

2. *Inertia of knowledge:* $\{\langle l,t\rangle \,|\mathit{knows}(l,t,n+1,b')$ is produced by (F3f)$\}$

To prove (B.5) for those $\langle l,t\rangle$ for which an atom $\mathit{knows}(l,t,n+1,b')$ is produced by logic programming rule (F3f), recall implication (12d):

$$\mathit{knows}(l,t,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftarrow \mathit{knows}(l,t,n,b') \in S_{\mathcal{D}}^{\boldsymbol{P}}$$

We substitute $\mathit{knows}(l,t,n+1,b')$ in (B.5) with the body of (12d) and obtain (B.6).

$$\mathit{knows}(l,t,n,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Rightarrow \mathit{eval}(\left\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}})\right\rangle) \models \langle l,t\rangle \tag{B.6}$$

with $t \le n$ and $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \cup \{\langle a,n\rangle \,|a \in \boldsymbol{A}_{n,b}\}$

By Lemma 2: $\forall l,t,n,b,b' : \left(\mathit{knows}(l,t,n,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \wedge (\exists l' : \mathit{sRes}(l',n,b,b') \in S_{\mathcal{D}}^{\boldsymbol{P}})\right) \Rightarrow b = b'$. Since the body of this implication holds under the case distinction (B.2), we only need to consider cases where $b = b'$.

$$\mathit{knows}(l,t,n,b) \in S_{\mathcal{D}}^{\boldsymbol{P}} \Rightarrow \mathit{eval}(\left\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}})\right\rangle) \models \langle l,t\rangle \tag{B.7}$$

with $t \le n$ and $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \cup \{\langle a,n\rangle \,|a \in \boldsymbol{A}_{n,b}\}$

By [8, Lemma 4]: $\langle l,t\rangle \in \boldsymbol{\kappa}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \Rightarrow \mathit{eval}(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}})\rangle) \models \langle l,t\rangle$

$$\mathit{knows}(l,t,n,b) \in S_{\mathcal{D}}^{\boldsymbol{P}} \Rightarrow \langle l,t\rangle \in \boldsymbol{\kappa}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \text{ with } t \le n. \tag{B.8}$$

By (10): $\boldsymbol{\kappa}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) = \{\langle l,t\rangle \,|\mathit{knows}(l,t,n,b) \in S_{\mathcal{D}}^{\boldsymbol{P}}\}$

We have shown for Case 1 (B.2) that the soundness Lemma (11) holds for those $\langle l,t\rangle$ for which $\mathit{knows}(l,t,n+1,b)$ is produced by inertia of knowledge (F3f).

3. *Sensing* $\{\langle l,t\rangle \,|\mathit{knows}(l,t,n+1,b)$ is produced by (F5k)$\}$

To prove (B.5) for those $\langle l,t\rangle$ for which an atom $\mathit{knows}(l,t,n{+}1,b')$ is produced by logic programming rule (F5k), we recall implication (12h):

$$\mathit{knows}(l,t,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftarrow (\mathit{sRes}(l,n,b,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \wedge t = n)$$

We substitute $\mathit{knows}(l,t,n+1,b')$ in (B.5) with the body of (12h) and obtain (B.9).

$$\forall l,t : \mathit{sRes}(l,n,b,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Rightarrow \mathit{eval}\left(\left\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \cup \langle l^s,t\rangle\right\rangle\right) \models \langle l,t\rangle \tag{B.9}$$

with $t \le n$ and $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \cup \{\langle a,n\rangle \,|a \in \boldsymbol{A}_{n,b}\}$

Recall that by (B.4), l^s is an arbitrary literal such that $\mathit{sRes}(l^s,n,b,b') \in S_{\mathcal{D}}^{\boldsymbol{P}}$. By Lemma 8:

$$\left(\mathit{sRes}(l,n,b,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \wedge \mathit{sRes}(l^s,n,b,b') \in S_{\mathcal{D}}^{\boldsymbol{P}}\right) \Rightarrow (l = l^s)$$

To this end, we consider $l = l^s$ from now.

$$\forall t : \mathit{sRes}(l^s,n,b,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Rightarrow \mathit{eval}\left(\left\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \cup \langle l^s,t\rangle\right\rangle\right) \models \langle l^s,t\rangle \tag{B.10}$$

with $t \le n$ and $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \cup \{\langle a,n\rangle \,|a \in \boldsymbol{A}_{n,b}\}$

By [8, Lemma 4]: $\mathit{eval}(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \cup \langle l^s,t\rangle\rangle) \models \langle l^s,t\rangle$

The base step is proven for knowledge about $\langle l,t\rangle$ generated by rule (F5k) (sensing) where $\exists l' : \mathit{sRes}(l',n,b,b') \in S_{\mathcal{D}}^{\boldsymbol{P}}$.

4. *Inheritance* $\{\langle l,t\rangle\,|knows(l,t,n+1,b')$ is produced by (F5m)$\}$

To prove (B.5) for those $\langle l,t\rangle$ for which an atom $knows(l,t,n{+}1,b')$ is produced by logic programming rule (F5m), recall the following implication (12i):

$$knows(l,t,n+1,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftarrow \big(\exists l' : \{sRes(l',n,b,b'), knows(l,t,n,b)\} \subseteq S_{\mathcal{D}}^{\boldsymbol{P}} \wedge n+1 \geq t\big)$$

We substitute $knows(l,t,n+1,b')$ in (B.5) with the body of (12h) and obtain (B.11).

$$\forall l,t:\ \Big(\big(\exists l' : \{sRes(l',n,b,b'), knows(l,t,n,b)\} \subseteq S_{\mathcal{D}}^{\boldsymbol{P}} \wedge n+1 \geq t\big) \Rightarrow \big(eval(\boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \cup \langle l^s,n\rangle) \models \langle l,t\rangle\,\big)\Big) \tag{B.11}$$

with $t \leq n$ and $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \cup \{\langle a,n\rangle\,|a \in \boldsymbol{A}_{n,b}\}$

With (10): $\forall\,\langle l,t\rangle : knows(l,t,n,b) \in S_{\mathcal{D}}^{\boldsymbol{P}} \Rightarrow \mathfrak{h}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \models \langle l,t\rangle$.

$$\forall l,t:\ \Big(\big(\exists l' : sRes(l',n,b,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \wedge \mathfrak{h}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \models \langle l,t\rangle\,\big) \Rightarrow \big(eval(\boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \cup \langle l^s,n\rangle) \models \langle l,t\rangle\,\big)\Big) \tag{B.12}$$

with $t \leq n$ and $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \cup \{\langle a,n\rangle\,|a \in \boldsymbol{A}_{n,b}\}$

By Lemma [8, 4]: $\forall\,\langle l,t\rangle : \langle l,t\rangle \in \boldsymbol{\kappa}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \Rightarrow eval(\boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}})) \models \langle l,t\rangle$. Hence, it is sufficient to show that (B.13) holds.

$$\forall l,t:\ \Big(\big(\exists l' : sRes(l',n,b,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \wedge \mathfrak{h}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \models \langle l,t\rangle\,\big) \Rightarrow \big(\,\langle l,t\rangle \in \boldsymbol{\kappa}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}})\big)\Big) \tag{B.13}$$

with $t \leq n$ and $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \cup \{\langle a,n\rangle\,|a \in \boldsymbol{A}_{n,b}\}$

With (10): $\mathfrak{h}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \models \langle l,t\rangle \Rightarrow \langle l,t\rangle \in \boldsymbol{\kappa}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}})$

The base step is proven for knowledge generated by rule (F5m) (inheritance).

Induction Steps for Case 1

1. *Forward inertia:* $\{\langle l,t\rangle \,| knows(l,t,n+1,b)$ is produced by (F3d)$\}$

To prove that (B.5) holds for those $knows(l,t,n+1,b')$ that are produced by (F3d), recall the following implication (12b):

$$\left(knows(l,t,n+1,b') \in S^{\boldsymbol{P}}_{\mathcal{D}}\right) \Leftarrow \left(\{knows(l,t-1,n+1,b'), kNotSet(\bar{l},t-1,n+1,b')\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}} \wedge t \leq n\right)$$

We substitute $knows(l,t,n+1,b')$ in (B.5) with the body of (12b) and obtain (B.14).

$$\{knows(l,t-1,n+1,b'), kNotSet(\bar{l},t-1,n+1,b')\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}} \wedge t \leq n \Rightarrow eval(\left\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l^s,n\rangle\right\rangle) \models \langle l,t\rangle \tag{B.14}$$

with $t \leq n$ and $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a,n\rangle \,| a \in \boldsymbol{A}_{n,b}\}$

We can eliminate $t \leq n$ since this is considered anyways. Further, recall that by the recursive definition of $eval$ [8, Eq. (15)]:

$$eval(\left\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l^s,n\rangle\right\rangle) = evalOnce(eval(\left\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l^s,n\rangle\right\rangle))$$

$$\left(\{knows(l,t-1,n+1,b'), kNotSet(\bar{l},t-1,n+1,b')\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}}\right) \Rightarrow \left(evalOnce(\mathfrak{h}') \models \langle l,t\rangle\right) \tag{B.15}$$

with $t \leq n$ and $\mathfrak{h}' = eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l^s,n\rangle\rangle)$ where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a,n\rangle \,| a \in \boldsymbol{A}_{n,b}\}$

Considering $evalOnce(\mathfrak{h}') = pd^{neg}(pd^{pos}(cause(back(fwd(\mathfrak{h}')))))$ [8, Eq. (14)] and its constituent functions [8, Eqs. (9), (10), (11), (12), (13)], in particular add_{fwd} [8, Eq. (9)], it follows that:

$$\langle l,t\rangle \in add_{fwd}(\mathfrak{h}') \Rightarrow evalOnce(\mathfrak{h}') \models \langle l,t\rangle$$

Hence, to show that (B.15) holds, it is sufficient to show that (B.16) holds:

$$\left(\{knows(l,t-1,n+1,b'), kNotSet(\bar{l},t-1,n+1,b')\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}}\right) \Rightarrow \left(\langle l,t\rangle \in add_{fwd}(\mathfrak{h}')\right) \tag{B.16}$$

with $t \leq n$ and $\mathfrak{h}' = eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l^s,n\rangle\rangle)$ where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a,n\rangle \,| a \in \boldsymbol{A}_{n,b}\}$

Consider Lemma 1:

$$\forall l,l',t,n,b',l^s : \left(\{kNotSet(l,t-1,n,b'), sRes(l^s,t-1,n,b')\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}} \wedge \left(knows(l',t-1,n,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow \langle l',t-1\rangle \in \boldsymbol{\kappa}'\right)\right)$$
$$\Rightarrow inertial(\bar{l},t-1,\mathfrak{h}')$$

Due to the induction hypothesis we may assume that $\left(knows(l',t-1,n,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow \langle l',t-1\rangle \in \boldsymbol{\kappa}'\right)$ is true, and hence we can substitute $kNotSet(l,t-1,n,b') \in S^{\boldsymbol{P}}_{\mathcal{D}}$ with $inertial(l,t-1,\mathfrak{h}')$.

$$\left(knows(l,t-1,n+1,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \wedge inertial(l,t-1,\mathfrak{h}')\right) \Rightarrow \left(\langle l,t\rangle \in add_{fwd}(\mathfrak{h}')\right) \tag{B.17}$$

with $t \leq n$ and $\mathfrak{h}' = eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l^s,n\rangle\rangle)$ where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a,n\rangle \,| a \in \boldsymbol{A}_{n,b}\}$

Again, the induction hypothesis allows us to assume that: $knows(l,t-1,n+1,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow \langle l,t-1\rangle \in \boldsymbol{\kappa}(\mathfrak{h}')$.

$$\left(\langle l,t-1\rangle \in \boldsymbol{\kappa}(\mathfrak{h}') \wedge inertial(l,t-1,\mathfrak{h}')\right) \Rightarrow \left(\langle l,t\rangle \in add_{fwd}(\mathfrak{h}')\right) \tag{B.18}$$

with $t \leq n$ and $\mathfrak{h}' = eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l',n\rangle\rangle)$ where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a,n\rangle \,| a \in \boldsymbol{A}_{n,b}\}$

Consider the definition of add_{fwd} [8, Eq. (9)]: $add_{fwd}(\mathfrak{h}') = \{\langle l,t\rangle \mid \langle l,t-1\rangle \in \boldsymbol{\kappa}(\mathfrak{h}') \wedge inertial(l,t-1,\mathfrak{h}') \wedge t \leq now(\mathfrak{h}')\}$. By definition of now [8, Eq. (4)], the definition of $\boldsymbol{A}_{n,b}$ (see Definition 2) and the fact that $\neg\exists \langle a,t\rangle \in \boldsymbol{\alpha}' : t > n$ (see (10b) it holds that $now(\mathfrak{h}') = n+1$

We have shown for Case 1 (B.2) that if knowledge is produced by forward inertia rule (F3d) the soundness Lemma (11) holds.

2. *Backward inertia:* $\{\langle l,t\rangle \,| knows(l,t,n+1,b)$ is produced by (F3e)$\}$
This case is analogous to the case of forward inertia.

3. *Causation:* $\{\langle l,t\rangle \,|knows(l,t,n+1,b)$ is produced by (F4c)$\}$

To prove (B.5) for those $\langle l,t\rangle$ for which an atom $knows(l,t,n+1,b')$ is produced by logic programming rule (F4c), recall the following implication (12e):

$$knows(l,t,n+1,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \Leftarrow kCause(l,t,n+1,b') \in S^{\boldsymbol{P}}_{\mathcal{D}}$$

To show that (B.5) holds for those $\langle l,t\rangle$ produced by LP rule (F4c) we prove (B.19).

$$kCause(l,t,n+1,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow eval(\Big\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l^s,n\rangle \Big\rangle) \models \langle l,t\rangle \tag{B.19}$$

with $t \leq n$ and $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a,n\rangle \,|a \in \boldsymbol{A}_{n,b}\}$

Due to the inductive definition of $eval$ [8, Eq. (15)]: $eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l^s,n\rangle\rangle) =$ $evalOnce(eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l^s,n\rangle\rangle))$.

$$kCause(l,t,n+1,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow evalOnce(\mathfrak{h}') \models \langle l,t\rangle \tag{B.20}$$

with $t \leq n$ and $\mathfrak{h}' = eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l^s,n\rangle\rangle)$ where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a,n\rangle \,|a \in \boldsymbol{A}_{n,b}\}$

Considering $evalOnce(\mathfrak{h}') = pd^{neg}(pd^{pos}(cause(back(fwd(\mathfrak{h}')))))$ [8, Eq. (14)] and its constituent functions [8, Eqs. (9), (10), (11), (12), (13)] it follows that $\langle l,t\rangle \in add_{cause}(\mathfrak{h}') \Rightarrow evalOnce(\mathfrak{h}') \models \langle l,t\rangle$. Hence, to show that (B.20) holds, it is sufficient to prove (B.21):

$$kCause(l,t,n+1,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow \langle l,t\rangle \in add_{cause}(\mathfrak{h}') \tag{B.21}$$

with $t \leq n$ and $\mathfrak{h}' = eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l^s,n\rangle\rangle)$ where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a,n\rangle \,|a \in \boldsymbol{A}_{n,b}\}$

Since $kCause/4$ atoms are only produced by LP rules generated by translation rule (T4a) we have the following biimplication:

$$\begin{aligned} kCause(l,t,n+1,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \;\Leftrightarrow\; \exists ep :\; & \big(e(ep) = l \wedge c(ep) = \{l_1^c,\ldots,l_k^c\} \wedge apply(ep,t-1,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \wedge n \geq t \wedge \\ & \{knows(l_1^c,t-1,n+1,b'),\ldots,knows(l_k^c,t-1,n+1,b')\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}}\big) \end{aligned} \tag{B.22}$$

$$\begin{aligned} & \exists ep : \big(e(ep) = l \wedge c(ep) = \{l_1^c,\ldots,l_k^c\} \wedge apply(ep,t-1,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \wedge n \geq t \wedge \\ & \qquad \{knows(l_1^c,t-1,n+1,b'),\ldots,knows(l_k^c,t-1,n+1,b')\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}}\big) \\ & \Rightarrow \langle l,t\rangle \in add_{cause}(\mathfrak{h}') \end{aligned} \tag{B.23}$$

with $t \leq n$ and $\mathfrak{h}' = eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l^s,n\rangle\rangle)$ where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a,n\rangle \,|a \in \boldsymbol{A}_{n,b}\}$

By the induction hypothesis:

$$\Big(\{knows(l_1^c,t-1,n+1,b'),\ldots,knows(l_k^c,t-1,n+1,b')\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}}\Big) \Rightarrow \big(\{\langle l_1^c,t-1\rangle,\ldots,\langle l_n^c,t-1\rangle\} \subseteq \boldsymbol{\kappa}(\mathfrak{h}')\big)$$

$$\begin{aligned} & \exists ep : \big(e(ep) = l \wedge c(ep) = \{l_1^c,\ldots,l_k^c\} \wedge apply(ep,t-1,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \wedge n \geq t \wedge \\ & \qquad \{\langle l_1^c,t-1\rangle,\ldots,\langle l_n^c,t-1\rangle\} \subseteq \boldsymbol{\kappa}(\mathfrak{h}')\big) \\ & \Rightarrow \langle l,t\rangle \in add_{cause}(\mathfrak{h}') \end{aligned} \tag{B.24}$$

with $t \leq n$ and $\mathfrak{h}' = eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l^s,n\rangle\rangle)$ where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a,n\rangle \,|a \in \boldsymbol{A}_{n,b}\}$

By Lemma 5: $(apply(ep,t-1,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \wedge sRes(l^s,n,b,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \wedge t-1 \leq n) \Rightarrow apply(ep,t-1,b)$
By Lemma 4: $apply(ep,t-1,b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow \langle ep,t-1\rangle \in \boldsymbol{\epsilon}(\mathfrak{h}')$

$$\begin{aligned} & \exists ep : \big(e(ep) = l \wedge c(ep) = \{l_1^c,\ldots,l_k^c\} \wedge \langle ep,t-1\rangle \in \boldsymbol{\epsilon}(\mathfrak{h}') \wedge n \geq t \wedge \\ & \qquad \{\langle l_1^c,t-1\rangle,\ldots,\langle l_n^c,t-1\rangle\} \subseteq \boldsymbol{\kappa}(\mathfrak{h}')\big) \\ & \Rightarrow \langle l,t\rangle \in add_{cause}(\mathfrak{h}') \end{aligned} \tag{B.25}$$

with $t \leq n$ and $\mathfrak{h}' = eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l^s,n\rangle\rangle)$ where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a,n\rangle \,|a \in \boldsymbol{A}_{n,b}\}$

To see that (B.25) holds, consider the definition of add_{cause} [8, Eq. (11)]:

$$add_{cause}(\mathfrak{h}') = \{\langle l,t\rangle \,|\exists \langle ep,t-1\rangle \in \boldsymbol{\epsilon}(\mathfrak{h}') : \big(e(ep) = l \wedge c(ep) = \{l_1^c,\ldots,l_k^c\} \wedge \{\langle l_1^c,t-1\rangle,\ldots,\langle l_n^c,t-1\rangle\} \subseteq \boldsymbol{\kappa}(\mathfrak{h}')\big)\}$$

We have shown for Case 2 (B.2) that if knowledge is produced by causation rule (F4c) the soundness Lemma (11) holds.

4. *Positive postdiction:* $\{\langle l,t\rangle \,|knows(l,t,n+1,b)$ is produced by (F4d)$\}$
   This case is analogous to the case of causation.

5. *Negative postdiction:* $\{\langle l,t\rangle \,|knows(l,t,n+1,b)$ is produced by (F4e)$\}$
   This case is analogous to the case of causation.

Case 2: $\neg\exists l' : (sRes(l',n,b,b') \in S_{\mathcal{D}}^{\boldsymbol{P}})$ *(No Sensing Result)*
To prove that (B.1) holds if no sensing results are obtained we consider cases where (B.26) holds.

$$\neg\exists l' : (sRes(l',n,b,b') \in S_{\mathcal{D}}^{\boldsymbol{P}}) \tag{B.26}$$

The proof for this case is very similar to the proof for Case 1. The difference is that no knowledge is produced by sensing and inheritance, and that $hasChild(n,b,b',S_{\mathcal{D}}^{\boldsymbol{P}})$ is always true since $b=b'$.

Appendix B.2. Auxiliary Lemmata Related to Knowledge

Appendix B.2.1. Inertia

The following Lemma is required in the induction step for proving soundness of forward inertia (B.14) in Appendix B.1.

Lemma 1 (Soundness for inertia in induction step).

$$\begin{aligned}\forall l,l',t,n,b',l^s : &\Big(\{kNotSet(l,t,n,b'), sRes(l^s,t,n,b')\} \subseteq S_{\mathcal{D}}^{\boldsymbol{P}} \wedge \big(knows(l',t,n,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Rightarrow \langle l',t\rangle \in \boldsymbol{\kappa}'\big)\Big) \\ &\Rightarrow inertial(\bar{l},t,\mathfrak{h}')\end{aligned} \tag{B.27}$$

with $t \leq n$ *and* $\mathfrak{h}' = eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \cup \langle l^s,n\rangle\rangle)$ *where* $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}) \cup \{\langle a,n\rangle \,|a \in \boldsymbol{A}_{n,b}\}$.

Proof:
Recall the following three $\mathcal{HPX}$-LP rules that affect $kNotSet/4$ atoms:

$$kNotSet(L,T,N,B) \leftarrow not\ kMaySet(L,T,B), uBr(N,B), s(T), literal(L). \tag{F3a}$$

$$kMaySet(L,T,B) \leftarrow apply(EP,T,B), hasEff(EP,L) \tag{F3b}$$

$$\begin{aligned}kNotSet(L,T,N,B) \leftarrow &apply(EP,T,B), hasCond(EP,L'), hasEff(EP,L), \\ &knows(\overline{L'},T,N,B), complement(L',\overline{L'}), N \geq T.\end{aligned} \tag{F3c}$$

Since (F3a) and (F3c) are the only rules in the LP with $kNotSet/4$ in their head, it holds that:

$$\begin{aligned}kNotSet(l,t,n,b) \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftrightarrow &\Big(\big(kMaySet(l,t,b) \notin S_{\mathcal{D}}^{\boldsymbol{P}} \wedge \{uBr(n,b), s(t), literal(l)\} \subseteq S_{\mathcal{D}}^{\boldsymbol{P}}\big) \\ &\vee \big(\exists ep, l^c : \big(apply(ep,t,b), hasCond(ep,l^c), hasEff(ep,l), knows(\overline{l^c},t,n,b) \subseteq S_{\mathcal{D}}^{\boldsymbol{P}} \wedge n \geq t\big)\big)\Big)\end{aligned} \tag{B.28}$$

We make a case distinction according to (B.28) and consider both possibilities that can trigger an atom $kNotSet(l,t,n,b)$.

1. $kNotSet/4$ generated by (F3a). In this case (B.29) holds and we prove (B.30)

$$\big(kMaySet(l,t,b) \notin S^{\boldsymbol{P}}_{\mathcal{D}} \wedge \{uBr(n,b), s(t), literal(l)\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}}\big) \tag{B.29}$$

$$\forall l, l', t, n, b', l^s : \Big(sRes(l^s, t, n, b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \wedge \big(kMaySet(l,t,b) \notin S^{\boldsymbol{P}}_{\mathcal{D}} \wedge \{uBr(n,b), s(t), literal(l)\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}}\big) \wedge \big(knows(l', t, n, b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow \langle l', t\rangle \in \boldsymbol{\kappa}'\big)\Big) \Rightarrow inertial(\bar{l}, t, \mathfrak{h}') \tag{B.30}$$

with $t \leq n$ and $\mathfrak{h}' = eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n, b, S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l^s, n\rangle\rangle)$ where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n, b, S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a, n\rangle \,| a \in \boldsymbol{A}_{n,b}\}$.

Consider rule (F3b). This is the only rule in the LP with $kMaySet/3$ in the head and therefore it holds that:

$$kMaySet(l,t,b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Leftrightarrow \exists ep : (\{apply(ep, t, b), hasEff(ep, l)\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}}) \tag{B.31}$$

With rewrite (B.31) as $kMaySet(l,t,b) \notin S^{\boldsymbol{P}}_{\mathcal{D}} \Leftrightarrow \forall ep : (apply(ep,t,b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow hasEff(ep,l) \notin S^{\boldsymbol{P}}_{\mathcal{D}})$ and make the following substitution:

$$\forall l, l', t, n, b', l^s : \Big(sRes(l^s, t, n, b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \wedge \big(\forall ep : (apply(ep,t,b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow hasEff(ep,l) \notin S^{\boldsymbol{P}}_{\mathcal{D}})\big) \wedge \big(\{uBr(n,b), s(t), literal(l)\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}}\big) \wedge \big(knows(l', t, n, b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow \langle l', t\rangle \in \boldsymbol{\kappa}'\big)\Big) \Rightarrow inertial(\bar{l}, t, \mathfrak{h}') \tag{B.32}$$

with $t \leq n$ and $\mathfrak{h}' = eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n, b, S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l^s, n\rangle\rangle)$ where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n, b, S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a, n\rangle \,| a \in \boldsymbol{A}_{n,b}\}$.

Consider the definition of $inertial$ [8, Eq. (8)]: $inertial(\bar{l}, t, \mathfrak{h}') \Leftrightarrow \forall \langle ep, t\rangle \in \boldsymbol{\epsilon}(\mathfrak{h}') : (e(ep) = l) \Rightarrow \big(\exists l^c \in c(ep) : \langle \overline{l^c}, t\rangle \in \boldsymbol{\kappa}'\big)$.

This proves that (B.30) holds.

2. $kNotSet/4$ generated by (F3c). In this case (B.33) holds and we prove (B.34)

$$\exists ep, l^c : (n \geq t \wedge \{apply(ep,t,b), hasCond(ep,l^c), hasEff(ep,l), knows(\overline{l^c},t,n,b)\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}}) \tag{B.33}$$

$$\begin{aligned}\forall l, l', t, n, b', l^s : \Big(& sRes(l^s,t,n,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \wedge \\ & \big(\exists ep, l' : (\{apply(ep,t,b'), hasEff(ep,l)\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}} \wedge n \geq t \wedge \{hasCond(ep,l'), knows(\overline{l'},t,n,b')\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}})\big) \wedge \\ & \big(knows(l',t,n,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow \langle l',t\rangle \in \boldsymbol{\kappa}'\big)\Big) \Rightarrow inertial(\bar{l},t,\mathfrak{h}')\end{aligned} \tag{B.34}$$

with $t \leq n$ and $\mathfrak{h}' = eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l^s,n\rangle\rangle)$ where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a,n\rangle \,|a \in \boldsymbol{A}_{n,b}\}$.

Recall LP rule (F2b) which restricts that two effect propositions with the same effect literal may not be applied concurrently:

$$\leftarrow apply(EP_1,T,B), hasEff(EP_1,L), apply(EP_2,T,B), hasEff(EP_2,L), EP_1 \neq EP_2, br(B), literal(L).$$

According to the semantics of ASP integrity constraints, the following holds due to (F2b):

$$\begin{aligned}\forall l, ep : & \left(\{apply(ep,t,b'), hasEff(ep,l)\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}}\right) \Rightarrow \\ & \left(\forall ep' \in S^{\boldsymbol{P}}_{\mathcal{D}} : \left(apply(ep',t,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow hasEff(ep,l) \notin S^{\boldsymbol{P}}_{\mathcal{D}}\right)\right)\end{aligned}$$

That is, only one effect proposition with the same effect literal l can be applied per state transition. Consequently, if an effect proposition with the effect literal l is applied, and the effect proposition has a condition literal l^c of which the complement is known to hold, then all effect propositions with the effect literal l that are applied have a condition literal l^c of which the complement is known. This is captured with the following equation:

$$\begin{aligned}&\big(\exists ep, l^c : (\{apply(ep,t,b'), hasEff(ep,l)\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}} \wedge \{hasCond(ep,l^c), knows(\overline{l^c},t,n,b')\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}})\big) \\ &\Rightarrow \big(\forall ep : \{\big((apply(ep,t,b'), hasEff(ep,l)\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}}\big) \Rightarrow \big(\exists l^c : \{hasCond(ep,l^c), knows(\overline{l^c},t,n,b')\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}}\big)\big)\big)\end{aligned}$$

$$\begin{aligned}\forall l, t, n, b', l^s : \Big(& sRes(l^s,t,n,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \wedge \\ & \big(\forall ep : \big((apply(ep,t,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \wedge hasEff(ep,l) \in S^{\boldsymbol{P}}_{\mathcal{D}}\big) \Rightarrow \big(\exists l^c : \{hasCond(ep,l^c), knows(\overline{l^c},t,n,b')\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}}\big)\big)\big) \wedge \\ & \big(knows(l',t,n,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow \langle l',t\rangle \in \boldsymbol{\kappa}'\big)\Big) \Rightarrow inertial(\bar{l},t,\mathfrak{h}')\end{aligned} \tag{B.35}$$

with $t \leq n$ and $\mathfrak{h}' = eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l^s,n\rangle\rangle)$ where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a,n\rangle \,|a \in \boldsymbol{A}_{n,b}\}$.

Consider Lemma 4: $\forall ep, t : apply(ep,t,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow \langle ep,t\rangle \in \boldsymbol{\epsilon}(\mathfrak{h}')$
By Lemma 9: $hasCond(ep,l^c) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow l^c \in c(ep)$, $hasEff(ep,l) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow l = e(ep)$, $hasCond(ep,l^c) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow l^c \in c(ep)$, $hasEff(ep,l) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow l = e(ep)$.
It further holds that: $\big(knows(\overline{l^c},t,n,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow \langle \overline{l^c},t\rangle \in \boldsymbol{\kappa}'\big)$. Hence, to show that (B.35) holds, it is sufficient to show that (B.36) holds.

$$\begin{aligned}\forall l, t, n, b', l^s : \Big(& sRes(l^s,t,n,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \wedge \\ & \big(\forall ep : \big((\langle ep,t\rangle \in \boldsymbol{\epsilon}(\mathfrak{h}') \wedge \Rightarrow l = e(ep)\big) \Rightarrow \big(\exists l^c : l^c \in c(ep) \wedge \langle \overline{l^c},t\rangle \in \boldsymbol{\kappa}'\big)\big)\big)\Big) \\ & \Rightarrow inertial(\bar{l},t,\mathfrak{h}')\end{aligned} \tag{B.36}$$

with $t \leq n$ and $\mathfrak{h}' = eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \langle l^s,n\rangle\rangle)$ where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a,n\rangle \,|a \in \boldsymbol{A}_{n,b}\}$.

Consider the definition of *inertial* [8, Eq. (8)]:

$$inertial(\bar{l},t,\mathfrak{h}') \Leftrightarrow (\forall \langle ep,t\rangle \in \boldsymbol{\epsilon}(\mathfrak{h}') : (e(ep) = l) \Rightarrow (\exists l^c \in c(ep) : \langle \overline{l^c},t\rangle \in \boldsymbol{\kappa}'))$$

We have shown that (B.34) holds.

We have proven the Lemma for both cases. □

Appendix B.2.2. No Knowledge in New Branches

The following Lemma is required for the proof of Theorem 1, in the base step of the inductive soundness proof.

Lemma 2 (Knowledge does not exist in new branches).

$$\forall l,t,n,b,b' : \left(knows(l,t,n,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \wedge (\exists l' : sRes(l',n,b,b') \in S_{\mathcal{D}}^{\boldsymbol{P}})\right) \Rightarrow b = b' \tag{B.37}$$

Proof Sketch:
Consider the following integrity constraint (F5i) in the domain independent theory of an $\mathcal{HPX}$-logic program which prevents that $sRes/4$ are produced in unused branches.

$$\leftarrow sRes(L,N,B,B'), uBr(N,B'), literal(L), neq(B,B') \tag{F5i}$$

It follows from the integrity constraint (F5i) that $(\exists l' : sRes(l',n,b,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \wedge b \neq b') \Rightarrow uBr(n,b') \notin S_{\mathcal{D}}^{\boldsymbol{P}}$.
Hence we can rewrite (B.37) as follows:

$$\forall l,t,n,b,b' : \left(knows(l,t,n,b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Rightarrow uBr(n,b') \in S_{\mathcal{D}}^{\boldsymbol{P}}\right) \tag{B.38}$$

Lemma 3 shows that (B.38) holds. This proves Lemma 2. □

Appendix B.2.3. No Knowledge in Unused Branches

The following Lemma 3 is required to prove Lemma 2.

Lemma 3 (Knowledge does not exist in unused branches).

$$\forall l,t,n,b : \left(knows(l,t,n,b) \in S_{\mathcal{D}}^{\boldsymbol{P}} \Rightarrow uBr(n,b) \in S_{\mathcal{D}}^{\boldsymbol{P}}\right) \tag{B.39}$$

Proof:
This To show that (B.39) is true, we go through all nine LP rules that generate a $knows/4$ atom. These are summarized as implications in Equation (12). We prove (B.39) by complete induction over the structure of these nine LP rules for l,t,n. For the base steps, we show that for some of the implications (12) it holds that (B.39) is true. For the induction steps we show that if (B.39) holds for certain triples $\langle l,t,n\rangle$ then it holds for other triples $\langle l',t',n'\rangle$ with $\langle l,t,n\rangle \neq \langle l',t',n'\rangle$. The induction is complete because we consider *all* implications in (12).
Base Steps

1. $$uBr(n,b) \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftarrow \left(t = 0 \wedge n = 0 \wedge b = 0 \wedge l \in \mathbf{VP}\right) \tag{B.40a}$$
   where **VP** is the set of literals for which a value proposition exists.

   This is true since by LP fact (F5a) it holds that $uBr(0,0) \in S_{\mathcal{D}}^{\boldsymbol{P}}$.

2. $$uBr(n,b) \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftarrow \left(sRes(l,n-1,b',b) \in S_{\mathcal{D}}^{\boldsymbol{P}}\right) \tag{B.40b}$$

   This is follows directly from LP rule (F5j).

3. $$uBr(n,b) \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftarrow \left(\{sRes(l',n-1,b',b), knows(l,t,n-1,b')\} \subseteq S_{\mathcal{D}}^{\boldsymbol{P}} \wedge n \geq t\right) \tag{B.40c}$$

   This is follows directly from LP rule (F5j).

Induction Steps

1.
$$uBr(n,b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Leftarrow \Big(\{knows(l,t-1,n,b), kNotSet(\bar{l},t-1,n,b)\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}} \wedge t \leq n\Big) \tag{B.40d}$$

This is true since by induction hypothesis it holds that $knows(l,t-1,n,b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow uBr(n,b) \in S^{\boldsymbol{P}}_{\mathcal{D}}$.

2.
$$uBr(n,b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Leftarrow \Big(\{knows(l,t+1,n,b), kNotSet(\bar{l},t-1,n,b)\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}} \wedge t \leq n\Big) \tag{B.40e}$$

This is true since by induction hypothesis it holds that $knows(l,t+1,n,b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow uBr(n,b) \in S^{\boldsymbol{P}}_{\mathcal{D}}$.

3.
$$uBr(n,b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Leftarrow \Big(knows(l,t,n-1,b) \in S^{\boldsymbol{P}}_{\mathcal{D}}\Big) \tag{B.40f}$$

This is true since by induction hypothesis it holds that $\{knows(l,t,n-1,b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow uBr(n-1,b) \in S^{\boldsymbol{P}}_{\mathcal{D}}$. By LP rule (F5c) it holds that $uBr(n-1,b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow uBr(n,b) \in S^{\boldsymbol{P}}_{\mathcal{D}}$.

4.
$$uBr(n,b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Leftarrow \Big(kCause(l,t,n,b) \in S^{\boldsymbol{P}}_{\mathcal{D}}\Big) \tag{B.40g}$$

By translation rule (T4a) and by considering that LP rules generated by (T4a) are the only LP rules with $kCause/4$ in the head it holds that

$$kCause(l,t,n,b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Leftrightarrow \Big(\exists ep : \big(e(ep) = l \wedge c(ep) = \{l^c_1, \ldots, l^c_k\} \wedge apply(ep,t-1,b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \\ \wedge\, n > t \wedge \{knows(l^c_1,t-1,n,b), \ldots, knows(l^c_k,t-1,n,b)\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}}\big)\Big)$$

Since we can assume by induction hypothesis that for all $i \in \{1,\ldots,k\}$: $knows(l^c_i,t-1,n,b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow uBr(n,b) \in S^{\boldsymbol{P}}_{\mathcal{D}}$ it must hold that (B.40g) is true.

5.
$$uBr(n,b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Leftarrow \Big(kCause(l,t,n,b) \in S^{\boldsymbol{P}}_{\mathcal{D}}\Big) \tag{B.40h}$$

This case is similar to (B.40g)

6.
$$uBr(n,b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Leftarrow \Big(kCause(l,t,n,b) \in S^{\boldsymbol{P}}_{\mathcal{D}}\Big) \tag{B.40i}$$

This case is similar to (B.40g)

□

Appendix B.3. Application of Effect Propositions

The following Lemma states that the application of effect propositions is sound. That is, whenever there exists an atom $apply(ep,t,b')$ in the stable model of an $\mathcal{HPX}$-logic program then there exists a pair $\langle ep,t\rangle$ in the Effect History $\boldsymbol{\epsilon}(\mathfrak{h})$ of a corresponding h-state.

Lemma 4 (Soundness of application of effect propositions).

$$\forall n,b,b' : hasChild(n,b,b',S^{\boldsymbol{P}}_{\mathcal{D}}) \Rightarrow \\ \big(\forall \mathfrak{h} \in \Psi(\boldsymbol{A}_{n,b}, \mathfrak{h}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}})) : \forall ep,t : (apply(ep,t,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \wedge t \leq n) \Rightarrow \langle ep,t\rangle \in \boldsymbol{\epsilon}(\mathfrak{h})\big) \tag{B.41}$$

Proof:
To prove (B.41) we make a case distinction to eliminate the $\forall b' : hasChild(n,b,b',S^{\boldsymbol{P}}_{\mathcal{D}})$-quantification. Specifically, we distinguish between (a) $\neg\exists l',b' : sRes(l',n,b,b') \in S^{\boldsymbol{P}}_{\mathcal{D}}$ and (b) $\exists l',b' : sRes(l',n,b,b') \in S^{\boldsymbol{P}}_{\mathcal{D}}$. Case (a) can be proven with simple substitutions and (b) requires a simple induction proof.

In both cases we argue that there are only two rules in the logic program with an $apply/3$ atom in the head. These are (F2a) and (F5n). Hence, if a stable model contains $apply/3$, then the body of one of (F2a), (F5n) must be compatible with the stable model. This is expressed with (B.42).

$$\forall n, b', ep, t : \; apply(ep, t, b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftrightarrow$$

$$\Big[\; \exists a : \Big(\{hasEP(a, ep), occ(a, t, b')\} \subset S_{\mathcal{D}}^{\boldsymbol{P}} \wedge n = t\Big) \tag{B.42a}$$

$$\vee \; \exists b, l : \Big(\{sRes(l, n, b, b'), apply(ep, t, b)\} \in S_{\mathcal{D}}^{\boldsymbol{P}} \wedge n \geq t\Big)\Big] \tag{B.42b}$$

Before making any case distinctions we simplify (B.41) as follows:

(B.41)

Transition function [8, Eq. (6)]:

$\Psi(\boldsymbol{A}_{n,b}, \mathfrak{h}(n, b, S_{\mathcal{D}}^{\boldsymbol{P}})) = \bigcup\limits_{k \in sense(\mathbf{A}, \mathfrak{h}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}))} eval(\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n, b, S_{\mathcal{D}}^{\boldsymbol{P}}) \cup k\rangle)$

where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n, b, S_{\mathcal{D}}^{\boldsymbol{P}}) \cup \{\langle a, t\rangle \, | a \in \boldsymbol{A}_{n,b} \wedge t = now(\mathfrak{h}(n, b, S_{\mathcal{D}}^{\boldsymbol{P}}))\}$.

It is further easy to see that $now(\mathfrak{h}(n, b, S_{\mathcal{D}}^{\boldsymbol{P}})) = n$ (see [8, Eq. (4)]).

$$\forall n, b, b' : hasChild(n, b, b', S_{\mathcal{D}}^{\boldsymbol{P}}) \Rightarrow$$

$$\Big(\forall \mathfrak{h} \in \bigcup_{k \in sense(\mathbf{A}, \mathfrak{h}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}))} eval(\Big\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n, b, S_{\mathcal{D}}^{\boldsymbol{P}}) \cup k\Big\rangle) : \forall ep, t : (apply(ep, t, b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \wedge t \leq n) \Rightarrow \langle ep, t\rangle \in \boldsymbol{\epsilon}(\mathfrak{h})\Big) \tag{B.43}$$

with $t \leq n$ and $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n, b, S_{\mathcal{D}}^{\boldsymbol{P}}) \cup \{\langle a, n\rangle \, | a \in \boldsymbol{A}_{n,b}\}$

By definition of the $eval$ function [8, Eq. (15)], re-evaluation does not affect the action history of an h-state. Formally:

$$\forall \mathfrak{h} \in \bigcup_{k \in sense(\mathbf{A}, \mathfrak{h}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}))} eval(\Big\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n, b, S_{\mathcal{D}}^{\boldsymbol{P}}) \cup k\Big\rangle) : \boldsymbol{\alpha}(\mathfrak{h}) = \boldsymbol{\alpha}'$$

By the definition of effect histories [8, Eq. (3)] it holds that:

$$\forall \mathfrak{h} \in \bigcup_{k \in sense(\mathbf{A}, \mathfrak{h}(n,b,S_{\mathcal{D}}^{\boldsymbol{P}}))} eval(\Big\langle \boldsymbol{\alpha}', \boldsymbol{\kappa}(n, b, S_{\mathcal{D}}^{\boldsymbol{P}}) \cup k\Big\rangle) : \boldsymbol{\epsilon}(\mathfrak{h}) = \boldsymbol{\epsilon}(\boldsymbol{\alpha}')$$

Hence we can eliminate the $\forall$ quantification over h-states $\mathfrak{h}$ and (B.43) is rewritten as follows:

$$\forall n, b, b' : hasChild(n, b, b', S_{\mathcal{D}}^{\boldsymbol{P}}) \Rightarrow \forall ep, t : (apply(ep, t, b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \wedge t \leq n) \Rightarrow \langle ep, t\rangle \in \boldsymbol{\epsilon}(\boldsymbol{\alpha}') \tag{B.44}$$

where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n, b, S_{\mathcal{D}}^{\boldsymbol{P}}) \cup \{\langle a, n\rangle \, | a \in \boldsymbol{A}_{n,b}\}$.

By [8, Eq. (3)]: $\boldsymbol{\epsilon}(\boldsymbol{\alpha}') = \{\langle ep, t\rangle \, | \exists \, \langle a, t\rangle \in \boldsymbol{\alpha}(\mathfrak{h}') : ep \in \mathcal{EP}^a\}$.

$$\forall n, b, b' : hasChild(n, b, b', S_{\mathcal{D}}^{\boldsymbol{P}}) \Rightarrow \forall ep, t : (apply(ep, t, b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \wedge t \leq n) \Rightarrow (\exists \, \langle a, t\rangle \in \boldsymbol{\alpha}' : ep \in \mathcal{EP}^a) \tag{B.45}$$

where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n, b, S_{\mathcal{D}}^{\boldsymbol{P}}) \cup \{\langle a, n\rangle \, | a \in \boldsymbol{A}_{n,b}\}$

To show that (B.45) holds we make a case distinction.

Case 1 - No Sensing Results

We consider the cases where (B.46) holds. That is, the following formulae are universally quantified over those n, b for which

(B.46) is true.

$$\neg\exists l', b' : sRes(l', n, b, b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \tag{B.46}$$

Recall (B.45):

$$\forall b' : hasChild(n, b, b', S^{\boldsymbol{P}}_{\mathcal{D}}) \Rightarrow \forall ep, t : (apply(ep, t, b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \wedge t \leq n) \Rightarrow (\exists \langle a, t\rangle \in \boldsymbol{\alpha}' : ep \in \mathcal{EP}^{a})$$

By (B.46) and (8):

$$((\neg\exists l' : sRes(l', n, b, b') \in S^{\boldsymbol{P}}_{\mathcal{D}}) \wedge hasChild(n, b, b')) \Rightarrow b = b'$$

That is, the following formulae are universally quantified over those n, b for which $\neg\exists l' : sRes(l', n, b, b')$ and $b = b'$ holds.

$$\forall ep, t : (apply(ep, t, b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \wedge t \leq n) \Rightarrow (\exists \langle a, t\rangle \in \boldsymbol{\alpha}' : ep \in \mathcal{EP}^{a}) \tag{B.47}$$

where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n, b, S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a, n\rangle \,| a \in \boldsymbol{A}_{n,b}\}$

There are two rules in an $\mathcal{HPX}$-logic program which have an atom $apply(ep, t, b)$ in the head, namely (F2a) and (F5n). We argue that $apply(ep, t, b)$ must be an atom in the stable model if the body of one of the rules is compatible with the stable model. This leads to the following case distinction:

1. Effect propositions triggered by action occurrence (F2a)

   Recall (B.47):

   $$\forall ep, t : (apply(ep, t, b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \wedge t \leq n) \Rightarrow (\exists \langle a, t\rangle \in \boldsymbol{\alpha}' : ep \in \mathcal{EP}^{a})$$

   where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n, b, S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a, n\rangle \,| a \in \boldsymbol{A}_{n,b}\}$

   Consider (B.42a):

   $$apply(ep, t, b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Leftarrow \exists a : (\{hasEP(a, ep), occ(a, t, b)\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}} \wedge n = t)$$

   The following formulae are universally quantified over those ep for which $\exists a : (\{hasEP(a, ep), occ(a, t, b)\} \subset S^{\boldsymbol{P}}_{\mathcal{D}} \wedge n = t)$ holds.

   $$(\exists a : \{hasEP(a, ep), occ(a, n, b)\} \subset S^{\boldsymbol{P}}_{\mathcal{D}}) \Rightarrow (\exists \langle a, n\rangle \in \boldsymbol{\alpha}' : ep \in \mathcal{EP}^{a}) \tag{B.48}$$

   where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n, b, S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a, n\rangle \,| a \in \boldsymbol{A}_{n,b}\}$

   Since $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n, b, S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a, n\rangle \,| a \in \boldsymbol{A}_{n,b}\}$ it is sufficient to show that (B.49) holds.

   $$(\exists a : \{hasEP(a, ep), occ(a, n, b)\} \subset S^{\boldsymbol{P}}_{\mathcal{D}}) \Rightarrow (\exists a \in \boldsymbol{A}_{n,b} : ep \in \mathcal{EP}^{a}) \tag{B.49}$$

   where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n, b, S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a, n\rangle \,| a \in \boldsymbol{A}_{n,b}\}$

   By Definition 2:

   $\boldsymbol{A}_{n,b} = \{a | occ(a, n, b) \in S^{\boldsymbol{P}}_{\mathcal{D}}\}$

   By Lemma 9:

   $\forall a, ep : (hasEP(a, ep) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Leftrightarrow ep \in \mathcal{EP}^{a})$

   We have proven that the application of effect propositions is sound if produced by rule (F2a) (application of EP triggered by action occurrence).

2. Effect propositions triggered by inheritance (F5n)

Recall (B.47):

$$\forall ep, t : (apply(ep,t,b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \land t \leq n) \Rightarrow (\exists \langle a,t \rangle \in \boldsymbol{\alpha}' : ep \in \mathcal{EP}^a)$$

where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a,n \rangle \, | a \in \boldsymbol{A}_{n,b}\}$

Consider (B.42b):
$apply(ep,t,b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \Leftarrow \exists l : (\{sRes(l,n,b,b), apply(ep,t,b)\} \in S^{\boldsymbol{P}}_{\mathcal{D}} \land n >= t)$

$\exists l : sRes(l,n,b,b) \in S^{\boldsymbol{P}}_{\mathcal{D}}$ contradicts the case distinction (B.46), hence no atom $apply(ep,t,b)$ is produced.

Case 2 - With Sensing Results: We consider the cases where (B.50) holds. That is, the following formulae are universally quantified over those n, b for which (B.50) is true.

$$\exists l' : sRes(l',n,b,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \tag{B.50}$$

The case distinction allows us to simplify (B.45). Consider the following substitutions:

Recall (B.45):

$$\forall n,b,b' : \quad hasChild(n,b,b',S^{\boldsymbol{P}}_{\mathcal{D}}) \Rightarrow$$
$$\forall ep, t : (apply(ep,t,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \land t \leq n) \Rightarrow (\exists \langle a,t \rangle \in \boldsymbol{\alpha}' : ep \in \mathcal{EP}^a)$$

By (B.50) and (8):

$$\forall b' : (\exists l' : sRes(l',n,b,b')) \Rightarrow hasChild(n,b,b') = true$$

The following formulae are universally quantified over those n, b for which (B.50) is true.

$$\forall ep, t, b' : (apply(ep,t,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \land t \leq n) \Rightarrow (\exists \langle a,t \rangle \in \boldsymbol{\alpha}' : ep \in \mathcal{EP}^a) \tag{B.51}$$

where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a,n \rangle \, | a \in \boldsymbol{A}_{n,b}\}$

To prove that (B.51) holds, we consider both rules of the $\mathcal{HPX}$-LP with an atom $apply(ep,t,b')$ in the head: (F2a) and (F5n). We argue that $apply(ep,t,b')$ must be an atom in the stable model if the body of one of the rules is compatible with the stable model. To this end, we perform induction over the structure of (B.42) for b'. For the base step show that (B.51) holds for those b' for which an atom $apply(ep,t,b')$ produced by rule (F2a). For the induction step we consider rule (F5n) which involves another $apply(ep,t,b'')$ atom: we argue that if (B.51) holds for a b' and if an atom $apply(ep,t,b'')$ is produced by rule (F5n), then (B.51) also holds for the b''. The induction is complete because rules (F2a) and (F5n) are the only rules with $apply(ep,t,b')$ atoms in the head.

1. Base Step: effect propositions triggered by action occurrence (F2a)
Consider (B.42a):

$$apply(ep,t,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \Leftarrow \exists a : \{hasEP(a,ep), occ(a,t,b')\} \subset S^{\boldsymbol{P}}_{\mathcal{D}} \land t = n \tag{B.52}$$

Due to Lemma 6 it holds that $\neg \exists b', l' : (b' \neq b \land sRes(l',n,b,b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \land occ(a,n,b') \in S^{\boldsymbol{P}}_{\mathcal{D}})$. Hence, we may only consider cases where $b' = b$. In this case, the proof is analogous to the soundness proof for effect propositions triggered by action occurrence in Case 1.

2. Induction Step: effect propositions triggered by inheritance (F5n)

Recall (B.47):

$$\forall ep, t : (apply(ep, t, b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \land t \leq n) \Rightarrow (\exists \langle a, t\rangle \in \boldsymbol{\alpha}' : ep \in \mathcal{EP}^a)$$

where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n, b, S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a, n\rangle \,| a \in \boldsymbol{A}_{n,b}\}$

Consider (B.42a):

$$apply(ep, t, b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \Leftarrow \exists l : (\{sRes(l, n, b, b'), apply(ep, t, b)\} \in S^{\boldsymbol{P}}_{\mathcal{D}} \land n >= t)$$

The following formulae are universally quantified for those ep for which $\exists l : (\{sRes(l, n, b, b'), apply(ep, t, b)\} \in S^{\boldsymbol{P}}_{\mathcal{D}} \land n \geq t)$ holds.

$$(\exists l : sRes(l, n, b, b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \land apply(ep, t, b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \land n \geq t) \Rightarrow (\exists \langle a, t\rangle \in \boldsymbol{\alpha}' : ep \in \mathcal{EP}^a) \tag{B.53}$$

where $\boldsymbol{\alpha}' = \boldsymbol{\alpha}(n, b, S^{\boldsymbol{P}}_{\mathcal{D}}) \cup \{\langle a, n\rangle \,| a \in \boldsymbol{A}_{n,b}\}$

Since we perform induction we assume that soundness holds for $apply(ep, t, b)$. That is,

$$(apply(ep, t, b) \in S^{\boldsymbol{P}}_{\mathcal{D}} \land t \leq n) \Rightarrow (\exists \langle a, t\rangle \in \boldsymbol{\alpha}' : ep \in \mathcal{EP}^a) \tag{B.54}$$

We have shown that (B.51) holds for $apply/3$ produced by the inheritance rule (F5n).

We have shown that the Lemma holds by proving that (B.41) holds for both cases, $\neg\exists b', l' : sRes(l', n, b, b') \in S^{\boldsymbol{P}}_{\mathcal{D}}$ and $\exists b', l' : sRes(l', n, b, b') \in S^{\boldsymbol{P}}_{\mathcal{D}}$. □

Lemma 5 (Branching of application of effect propositions).

$$\forall l, n, b, b', ep, t : (\{sRes(l', n, b, b'), apply(ep, t, b)\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}} \land t \leq n) \Leftrightarrow (\{sRes(l', n, b, b'), apply(ep, t, b')\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}} \land t \leq n) \tag{B.55}$$

Proof:
We distinguish two cases: The $\Rightarrow$ direction directly emerges from the inheritance rule (F5n). For the $\Leftarrow$ direction we consider two cases:

1. Consider that an atom $apply(ep, t, b')$ is produced by rule (F2a). In this case it must hold that $\{occ(a, n, b'), hasEP(a, ep)\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}} \land n = t$ and by Definition 2 it holds that $occ(a, n, b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow uBr(n, b') \in S^{\boldsymbol{P}}_{\mathcal{D}}$. However, considering that $sRes(l', n, b, b') \in S^{\boldsymbol{P}}_{\mathcal{D}}$ the integrity constraint (F5i) assures that $uBr(n, b') \notin S^{\boldsymbol{P}}_{\mathcal{D}}$ and leads to a contradiction. Hence $apply(ep, t, b')$ can not be produced by (F2a) if $sRes(l', n, b, b')S^{\boldsymbol{P}}_{\mathcal{D}}$.

2. Consider that an atom $apply(ep, t, b')$ is produced by rule (F5n). Then clearly it must hold that $apply(ep, t, b) \in S^{\boldsymbol{P}}_{\mathcal{D}}$.

Items 1. and 2. prove the Lemma □

Lemma 6 (Actions do not occur in new branches).

$$\forall n, b, a : \neg\exists b', l' : (b' \neq b \land sRes(l', n, b, b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \land occ(a, n, b') \in S^{\boldsymbol{P}}_{\mathcal{D}}) \tag{B.56}$$

Proof: Suppose the contrary is true, i.e. $\exists b', l' : (b' \neq b \land sRes(l', n, b, b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \land occ(a, n, b') \in S^{\boldsymbol{P}}_{\mathcal{D}})$. Then by Definition 2 it must hold that $uBr(n, b') \in S^{\boldsymbol{P}}_{\mathcal{D}}$. This again contradicts the integrity constraint (F5i), hence (B.56) must hold. □

Appendix B.4. Sensing Results

We prove soundness for sensing results: if at a node n, b an atom $sRes(l, n, b, b')$ is produced, then the $sense$-function [8, Eq. (7)] returns a pair $\langle l, t\rangle$. This is captured with Lemma 7.

Lemma 7 (Soundness for sensing results).

$$\forall l, n, b, b' : sRes(l, n, b, b') \in S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow sense(\boldsymbol{A}_{n,b}, \mathfrak{h}(n, b, S^{\boldsymbol{P}}_{\mathcal{D}})) = \{\langle l, n\rangle, \langle \bar{l}, n\rangle \tag{B.57}$$

$$\forall n, b, b' : (\neg\exists l : sRes(l, n, b, b') \in S^{\boldsymbol{P}}_{\mathcal{D}}) \Rightarrow (sense(\boldsymbol{A}_{n,b}, \mathfrak{h}(n, b, S^{\boldsymbol{P}}_{\mathcal{D}})) = \{\emptyset\}) \tag{B.58}$$

Proof:
Consider all rules in an $\mathcal{HPX}$-logic program with an $sRes/4$ atom in the head. These are:

$$sRes(F,N,B,B) \leftarrow occ(A,N,B), hasKP(A,F), not\ kw(F,N,N,B) \quad \text{(F5f)}$$
$$1\{sRes(neg(F),N,B,B') : neq(B,B')\}1 \leftarrow occ(A,N,B), hasKP(A,F), not\ kw(F,N,N,B) \quad \text{(F5g)}$$

We need the following auxiliary result. Consider rules (F5d),(F5e) which produce $kw/4$ atoms. According to the stable model semantics, since these rules are the only rules with $kw/4$ in their heads the following must hold:

$$\begin{aligned}&\forall f,t,n,b:\\&kw(f,t,n,b) \notin S^{\boldsymbol{P}}_{\mathcal{D}} \Rightarrow \{knows(f,t,n,b), knows(neg(f),t,n,b)\} \cap S^{\boldsymbol{P}}_{\mathcal{D}} = \emptyset\end{aligned} \tag{B.59}$$

To prove that (B.57) holds, we show that for both rules (F5f), (F5g) that if their body is compatible with the stable model $S^{\boldsymbol{P}}_{\mathcal{D}}$, then $\exists \langle l,n\rangle \in sense(\boldsymbol{A}_{n,b}, \mathfrak{h}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}))$ must hold. That is, to show that (B.57) holds we consider (B.59) and show that both (B.60) and (B.61) hold:

$$\begin{aligned}&\forall f,n,b:\exists a:\big(\{occ(a,n,b), hasKP(a,f)\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}} \wedge \{knows(f,n,n,b), knows(neg(f),n,n,b)\} \cap S^{\boldsymbol{P}}_{\mathcal{D}} = \emptyset\big)\\&\qquad \Rightarrow (sense(\boldsymbol{A}_{n,b}, \mathfrak{h}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}})) = \{\langle l,n\rangle, \langle \bar{l},n\rangle\})\end{aligned} \tag{B.60}$$

$$\begin{aligned}&\forall f,n,b:\exists a:\big(\{occ(a,n,b), hasKP(a,f)\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}} \wedge \{knows(f,n,n,b), knows(neg(f),n,n,b)\} \cap S^{\boldsymbol{P}}_{\mathcal{D}} = \emptyset\big)\\&\qquad \Rightarrow (sense(\boldsymbol{A}_{n,b}, \mathfrak{h}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}})) = \{\langle l,n\rangle, \langle \bar{l},n\rangle\})\end{aligned} \tag{B.61}$$

1. Positive sensing result (B.60)
   (B.60)

   With Definition 2 and Lemma 9:
   $\exists a : \big(\{occ(a,n,b), hasKP(a,f)\} \subseteq S^{\boldsymbol{P}}_{\mathcal{D}}\big)$
   $\Rightarrow (\exists a \in \boldsymbol{A}_{n,b} : \mathcal{KP}^a = f)$

   $$\begin{aligned}&\forall f,n,b:\exists a \in \boldsymbol{A}_{n,b}:\big(\mathcal{KP}^a = f \wedge \{knows(f,n,n,b), knows(neg(f),n,n,b)\} \cap S^{\boldsymbol{P}}_{\mathcal{D}} = \emptyset\big)\\&\qquad \Rightarrow (sense(\boldsymbol{A}_{n,b}, \mathfrak{h}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}})) = \{\langle f,n\rangle, \langle \neg f,n\rangle\})\end{aligned} \tag{B.62}$$

   By (10): $\boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) = \{\langle l,t\rangle \mid knows(l,t,n,b) \in S^{\boldsymbol{P}}_{\mathcal{D}}\}$

   $\{knows(f,n,n,b), knows(neg(f),n,n,b)\} \cap S^{\boldsymbol{P}}_{\mathcal{D}} = \emptyset \Rightarrow \{\langle \neg f,n\rangle, \langle f,n\rangle\} \cap \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) = \emptyset$

   $$\begin{aligned}&\forall f,n,b:\exists a \in \boldsymbol{A}_{n,b}:\mathcal{KP}^a = f \wedge \{\langle \neg f,n\rangle, \langle f,n\rangle\} \cap \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) = \emptyset\big)\\&\qquad \Rightarrow (sense(\boldsymbol{A}_{n,b}, \mathfrak{h}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}})) = \{\langle f,n\rangle, \langle \neg f,n\rangle\})\end{aligned} \tag{B.63}$$

   Consider [8, Eq. (7)], and according to now [8, Eq. (4)], (8) and (10): $now(\mathfrak{h}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}})) = n$. Further, $\boldsymbol{\kappa}(\mathfrak{h}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}})) = \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}})$.

   $$sense(\boldsymbol{A}_{n,b}, \mathfrak{h}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}})) = \begin{cases}\{\{\langle f,n\rangle\}, \{\langle \neg f,n\rangle\}\} & \text{if } \exists a \in \boldsymbol{A}_{n,b} : \mathcal{KP}^a = f \wedge \langle f,n\rangle \notin \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}}) \wedge \langle \neg f,n\rangle \notin \boldsymbol{\kappa}(n,b,S^{\boldsymbol{P}}_{\mathcal{D}})\\ \{\emptyset\} & \text{otherwise}\end{cases}$$

   It follows that (B.63) holds.

2. Negative sensing result (B.61)
   This is analogous to the case of the positive sensing result.

The proof for (B.58) is analogous to the proof for (B.57). □

Lemma 8 (Only one sensing result per branch).

$$\forall n, l, b, b', l' : ((sRes(l, n, b, b') \in S_{\mathcal{D}}^{\boldsymbol{P}} \wedge sRes(l', n, b, b') \in S_{\mathcal{D}}^{\boldsymbol{P}}) \Rightarrow l = l') \tag{B.64}$$

Proof Sketch:
This directly follows from the integrity constraint (F5h). □

Appendix B.5. Auxiliary Predicates

Lemma 9 (Soundness for auxiliary predicates). *Given the prerequisites described in Definition 2 the following holds:*

1. $hasEP(a, ep) \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftrightarrow ep \in \mathcal{EP}^a$
2. $hasEff(ep, l) \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftrightarrow e(ep) = l$
3. $hasCond(ep, l) \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftrightarrow l \in c(ep)$
4. $hasKP(a, f) \in S_{\mathcal{D}}^{\boldsymbol{P}} \Leftrightarrow \mathcal{KP}^a = f$

Proof:
This follows directly from observing that the auxiliary predicates $hasEP/2$, $hasEff/2$, $hasCond/2$, $hasKP/2$ only appear in a stable model $S_{\mathcal{D}}^{\boldsymbol{P}}$ if they are produced by translation rules (T3) – (T5). There are no other rules which have one of the auxiliary predicates in their head. □