Online appendix for the paper

Bridging Commonsense Reasoning and Probabilistic Planning via a Probabilistic Action Language

published in Theory and Practice of Logic Programming

Yi Wang[*], Shiqi Zhang[#], Joohyung Lee[*]
[*]Arizona State University, USA [#] SUNY Binghamton, USA

Appendix A Extended Review of Preliminaries

A.1 Review: Language $\mathrm{LP}^{\mathrm{MLN}}$

An $\mathrm{LP}^{\mathrm{MLN}}$ program is a finite set of weighted rules $w : R$ where R is a rule and w is a real number (in which case, the weighted rule is called *soft*) or α for denoting the infinite weight (in which case, the weighted rule is called *hard*). Throughout the paper, we assume that the language is propositional. Schematic variables can be introduced via grounding as usual in answer set programming.

For any $\mathrm{LP}^{\mathrm{MLN}}$ program Π and any interpretation I, $\overline{\Pi}$ denotes the usual (unweighted) ASP program obtained from Π by dropping the weights, and Π_I denotes the set of $w : R$ in Π such that $I \models R$.

In general, an $\mathrm{LP}^{\mathrm{MLN}}$ program may even have stable models that violate some hard rules, which encode definite knowledge. However, throughout the paper, we restrict attention to $\mathrm{LP}^{\mathrm{MLN}}$ programs whose stable models do not violate hard rules. More precisely, given a ground $\mathrm{LP}^{\mathrm{MLN}}$ program Π, $\mathrm{SM}[\Pi]$ denotes the set

$$\{I \mid I \text{ is a (deterministic) stable model of } \Pi_I \text{ that satisfies all hard rules in } \Pi\}.$$

The weight of an interpretation I, denoted $W_\Pi(I)$, is defined as

$$W_\Pi(I) = \begin{cases} exp\left(\sum\limits_{w:R\ \in\ \Pi_I} w\right) & \text{if } I \in \mathrm{SM}[\Pi]; \\ 0 & \text{otherwise,} \end{cases}$$

and the probability of I, denoted $P_\Pi(I)$, is defined as

$$P_\Pi(I) = \frac{W_\Pi(I)}{\sum\limits_{J \in \mathrm{SM}[\Pi]} W_\Pi(J)}.$$

A.2 Review: $\mathrm{DT\text{-}LP}^{\mathrm{MLN}}$

We extend the syntax and the semantics of $\mathrm{LP}^{\mathrm{MLN}}$ to $\mathrm{DT\text{-}LP}^{\mathrm{MLN}}$ by introducing atoms of the form

$$\mathtt{utility}(u, \mathbf{t}) \tag{A1}$$

where u is a real number, and $\mathbf{t}$ is an arbitrary list of terms. These atoms can only occur in the head of hard rules of the form

$$\alpha : \mathtt{utility}(u, \mathbf{t}) \leftarrow \textit{Body} \tag{A2}$$

where *Body* is a list of literals. We call these rules *utility rules*.

The weight and the probability of an interpretation are defined the same as in LP$^{\rm MLN}$. The *utility* of an interpretation I under Π is defined as

$$U_{\Pi}(I) = \sum_{\texttt{utility}(u,\texttt{t}) \in I} u.$$

The *expected utility* of a proposition A is defined as

$$E[U_{\Pi}(A)] = \sum_{I \models A} U_{\Pi}(I) \times P_{\Pi}(I \mid A). \tag{A3}$$

A.3 Review: Multi-Valued Probabilistic Programs

Multi-valued probabilistic programs (Lee and Wang 2016) are a simple fragment of LP$^{\rm MLN}$ that allows us to represent probability more naturally.

We assume that the propositional signature σ is constructed from "constants" and their "values." A *constant* c is a symbol that is associated with a finite set *Dom*(c), called the *domain*. The signature σ is constructed from a finite set of constants, consisting of atoms $c = v$ [7] for every constant c and every element v in *Dom*(c). If the domain of c is $\{\text{FALSE}, \text{TRUE}\}$ then we say that c is *Boolean*, and abbreviate $c\!=\!\text{TRUE}$ as c and $c\!=\!\text{FALSE}$ as $\sim\!c$.

We assume that constants are divided into *probabilistic* constants and *non-probabilistic* constants. A multi-valued probabilistic program $\mathbf{\Pi}$ is a tuple $\langle PF, \Pi \rangle$, where

- *PF* contains *probabilistic constant declarations* of the following form:

$$p_1 :: \ c\!=\!v_1 \mid \cdots \mid p_n :: \ c\!=\!v_n \tag{A4}$$

  one for each probabilistic constant c, where $\{v_1, \dots, v_n\} = Dom(c)$, $v_i \neq v_j$, $0 \leq p_1, \dots, p_n \leq 1$ and $\sum_{i=1}^{n} p_i = 1$. We use $M_{\mathbf{\Pi}}(c = v_i)$ to denote p_i. In other words, *PF* describes the probability distribution over each "random variable" c.
- Π is a set of rules such that the head contains no probabilistic constants.

The semantics of such a program $\mathbf{\Pi}$ is defined as a shorthand for LP$^{\rm MLN}$ program $T(\mathbf{\Pi})$ of the same signature as follows.

- For each probabilistic constant declaration (A4), $T(\mathbf{\Pi})$ contains, for each $i = 1, \dots, n$, (i) $ln(p_i) : c\!=\!v_i$ if $0 < p_i < 1$; (ii) $\alpha : \ c\!=\!v_i$ if $p_i = 1$; (iii) $\alpha : \ \bot \leftarrow c\!=\!v_i$ if $p_i = 0$.
- For each rule *Head* $\leftarrow$ *Body* in Π, $T(\mathbf{\Pi})$ contains $\alpha :$ *Head* $\leftarrow$ *Body*.
- For each constant c, $T(\mathbf{\Pi})$ contains the uniqueness of value constraints

$$\alpha : \quad \bot \leftarrow c\!=\!v_1 \wedge c = v_2 \tag{A5}$$

  for all $v_1, v_2 \in Dom(c)$ such that $v_1 \neq v_2$, and the existence of value constraint

$$\alpha : \quad \bot \leftarrow \neg \bigvee_{v \in Dom(c)} c\!=\!v \ . \tag{A6}$$

In the presence of the constraints (A5) and (A6), assuming $T(\mathbf{\Pi})$ has at least one (probabilistic) stable model that satisfies all the hard rules, a (probabilistic) stable model I satisfies $c = v$ for exactly one value v, so we may identify I with the value assignment that assigns v to c.

[7] Note that here "=" is just a part of the symbol for propositional atoms, and is not equality in first-order logic.

A.4 Review: Action Language $p\mathcal{BC}+$ with Utility

Syntax of $p\mathcal{BC}+$

We assume a propositional signature σ as defined in Section A.3. We further assume that the signature of an action description is divided into four groups: *fluent constants*, *action constants*, *pf (probability fact) constants* and *initpf (initial probability fact) constants*. Fluent constants are further divided into *regular* and *statically determined*. The domain of every action constant is Boolean. A *fluent formula* is a formula such that all constants occurring in it are fluent constants.

The following definition of $p\mathcal{BC}$+ is based on the definition of $\mathcal{BC}$+ language from (Babb and Lee 2015).

A *static law* is an expression of the form

$$\textbf{caused } F \textbf{ if } G \tag{A7}$$

where F and G are fluent formulas.

A *fluent dynamic law* is an expression of the form

$$\textbf{caused } F \textbf{ if } G \textbf{ after } H \tag{A8}$$

where F and G are fluent formulas and H is a formula, provided that F does not contain statically determined constants and H does not contain initpf constants.

A *pf constant declaration* is an expression of the form

$$\textbf{caused } c = \{v_1 : p_1, \dots, v_n : p_n\} \tag{A9}$$

where c is a pf constant with domain $\{v_1, \dots, v_n\}$, $0 < p_i < 1$ for each $i \in \{1, \dots, n\}$[8], and $p_1 + \dots + p_n = 1$. In other words, (A9) describes the probability distribution of c.

An *initpf constant declaration* is an expression of the form (A9) where c is an initpf constant.

An *initial static law* is an expression of the form

$$\textbf{initially } F \textbf{ if } G \tag{A10}$$

where F is a fluent constant and G is a formula that contains neither action constants nor pf constants.

A *causal law* is a static law, a fluent dynamic law, a pf constant declaration, an initpf constant declaration, or an initial static law. An *action description* is a finite set of causal laws.

We use σ^{fl} to denote the set of fluent constants, σ^{act} to denote the set of action constants, σ^{pf} to denote the set of pf constants, and σ^{initpf} to denote the set of initpf constants. For any signature σ' and any $i \in \{0, \dots, m\}$, we use $i : \sigma'$ to denote the set $\{i : a \mid a \in \sigma'\}$.

By $i : F$ we denote the result of inserting $i :$ in front of every occurrence of every constant in formula F. This notation is straightforwardly extended when F is a set of formulas.

Semantics of $p\mathcal{BC}+$

Given a non-negative integer m denoting the maximum length of histories, the semantics of an action description D in $p\mathcal{BC}$+ is defined by a reduction to multi-valued probabilistic program $Tr(D, m)$, which is the union of two subprograms D_m and D_{init} as defined below.

[8] We require $0 < p_i < 1$ for each $i \in \{1, \dots, n\}$ for the sake of simplicity. On the other hand, if $p_i = 0$ or $p_i = 1$ for some i, that means either v_i can be removed from the domain of c or there is not really a need to introduce c as a pf constant. So this assumption does not really sacrifice expressivity.

For an action description D of a signature σ, we define a sequence of multi-valued probabilistic program $D_0, D_1, \dots,$ so that the stable models of D_m can be identified with the paths in the transition system described by D.

The signature σ_m of D_m consists of atoms of the form $i : c = v$ such that

- for each fluent constant c of D, $i \in \{0, \dots, m\}$ and $v \in Dom(c)$,
- for each action constant or pf constant c of D, $i \in \{0, \dots, m-1\}$ and $v \in Dom(c)$.

For $x \in \{act, fl, pf\}$, we use σ^x_m to denote the subset of σ_m

$$\{i : c = v \mid i : c = v \in \sigma_m \text{ and } c \in \sigma^x\}.$$

For $i \in \{0, \dots, m\}$, we use $i : \sigma^x$ to denote the subset of σ^x_m

$$\{i : c = v \mid i : c = v \in \sigma^x_m\}.$$

We define D_m to be the multi-valued probabilistic program $\langle PF, \Pi\rangle$, where Π is the conjunction of

$$i : F \leftarrow i : G \tag{A11}$$

for every static law (A7) in D and every $i \in \{0, \dots, m\}$,

$$i{+}1 : F \leftarrow (i{+}1 : G) \wedge (i : H) \tag{A12}$$

for every fluent dynamic law (A8) in D and every $i \in \{0, \dots, m-1\}$,

$$\{0{:}c = v\}^{\rm ch} \tag{A13}$$

for every regular fluent constant c and every $v \in Dom(c)$,

$$\{i : c = \text{TRUE}\}^{\rm ch}, \quad \{i : c = \text{FALSE}\}^{\rm ch} \tag{A14}$$

for every action constant c, and PF consists of

$$p_1 :: \ i : pf = v_1 \mid \cdots \mid p_n :: \ i : pf = v_n \tag{A15}$$

($i = 0, \dots, m-1$) for each pf constant declaration (A9) in D that describes the probability distribution of *pf*.

In addition, we define the program D_{init}, whose signature is $0{:}\sigma^{initpf} \cup 0{:}\sigma^{fl}$. D_{init} is the multi-valued probabilistic program

$$D_{init} = \langle PF^{init}, \Pi^{init}\rangle$$

where Π^{init} consists of the rule

$$\bot \leftarrow \neg(0{:}F) \wedge 0{:}G$$

for each initial static law (A10), and PF^{init} consists of

$$p_1 :: \ 0{:}pf = v_1 \ \mid \ \dots \ \mid \ p_n :: \ 0{:}pf = v_n$$

for each initpf constant declaration (A9).

We define $Tr(D, m)$ to be the union of the two multi-valued probabilistic program $\langle PF \cup PF^{init}, \Pi \cup \Pi^{init}\rangle$.

For any LP$^{\rm MLN}$ program Π of signature σ and a value assignment I to a subset σ' of σ, we say I is a *residual (probabilistic) stable model* of Π if there exists a value assignment J to $\sigma \setminus \sigma'$ such that $I \cup J$ is a (probabilistic) stable model of Π.

For any value assignment I to constants in σ, by $i : I$ we denote the value assignment to constants in $i\!:\!\sigma$ so that $i\!:\!I \models (i\!:\!c) = v$ iff $I \models c = v$.

We define a *state* as an interpretation I^{fl} of σ^{fl} such that $0 : I^{fl}$ is a residual (probabilistic) stable model of D_0. A *transition* of D is a triple $\langle s, e, s'\rangle$ where s and s' are interpretations of σ^{fl} and e is a an interpretation of σ^{act} such that $0 : s \cup 0 : e \cup 1 : s'$ is a residual stable model of D_1. A *pf-transition* of D is a pair $(\langle s, e, s'\rangle, pf)$, where pf is a value assignment to σ^{pf} such that $0\!:\!s \cup 0\!:\!e \cup 1 : s' \cup 0\!:\!pf$ is a stable model of D_1.

A *probabilistic transition system* $T(D)$ represented by a probabilistic action description D is a labeled directed graph such that the vertices are the states of D, and the edges are obtained from the transitions of D: for every transition $\langle s, e, s'\rangle$ of D, an edge labeled $e : p$ goes from s to s', where $p = Pr_{D_1}(1\!:\!s' \mid 0\!:\!s, 0\!:\!e)$. The number p is called the *transition probability* of $\langle s, e, s'\rangle$.

The soundness of the definition of a probabilistic transition system relies on the following proposition.

Proposition 1
For any transition $\langle s, e, s'\rangle$, s and s' are states.

We make the following simplifying assumptions on action descriptions:

1. **No concurrent execution of actions**: For all transitions $\langle s, e, s'\rangle$, we have $e \models a = \text{TRUE}$ for at most one action constant a;
2. **Nondeterministic transitions are determined by pf constants**: For any state s, any value assignment e of σ^{act}, and any value assignment pf of σ^{pf}, there exists exactly one state s' such that $(\langle s, e, s'\rangle, pf)$ is a pf-transition;
3. **Nondeterminism on initial states are determined by initpf constants**: For any value assignment pf_{init} of σ^{initpf}, there exists exactly one value assignment fl of σ^{fl} such that $0\!:\!pf_{init} \cup 0\!:\!fl$ is a stable model of $D_{init} \cup D_0$.

For any state s, any value assignment e of σ^{act} such that at most one action is true, and any value assignment pf of σ^{pf}, we use $\phi(s, e, pf)$ to denote the state s' such that $(\langle s, a, s'\rangle, pf)$ is a pf-transition (According to Assumption 2, such s' must be unique). For any interpretation I, $i \in \{0, \dots, m\}$ and any subset σ' of σ, we use $I|_{i:\sigma'}$ to denote the value assignment of I to atoms in $i : \sigma'$. Given any value assignment TC of $0\!:\!\sigma^{initpf} \cup \sigma^{pf}_m$ and a value assignment A of σ^{act}_m, we construct an interpretation $I_{TC\cup A}$ of $Tr(D, m)$ that satisfies $TC \cup A$ as follows:

- For all atoms p in $\sigma^{pf}_m \cup 0\!:\!\sigma^{initpf}$, we have $I_{TC\cup A}(p) = TC(p)$;
- For all atoms p in σ^{act}_m, we have $I_{TC\cup A}(p) = A(p)$;
- $(I_{TC\cup A})|_{0:\sigma^{fl}}$ is the assignment such that $(I_{TC\cup A})|_{0:\sigma^{fl} \cup 0:\sigma^{initpf}}$ is a stable model of $D_{init} \cup D_0$.
- For each $i \in \{1, \dots, m\}$,

$$(I_{TC\cup A})|_{i:\sigma^{fl}} = \phi((I_{TC\cup A})|_{(i-1):\sigma^{fl}}, (I_{TC\cup A})|_{(i-1):\sigma^{act}}, (I_{TC\cup A})|_{(i-1):\sigma^{pf}}).$$

By Assumptions 2 and 3, the above construction produces a unique interpretation.

It can be seen that in the multi-valued probabilistic program $Tr(D, m)$ translated from D, the probabilistic constants are $0\!:\!\sigma^{initpf} \cup \sigma^{pf}_m$. We thus call the value assignment of an interpretation I on $0\!:\!\sigma^{initpf} \cup \sigma^{pf}_m$ the *total choice* of I. The following theorem asserts that the probability of a stable model under $Tr(D, m)$ can be computed by simply dividing the probability of the total choice associated with the stable model by the number of choice of actions.

Theorem 1
For any value assignment TC of $0 : \sigma^{initpf} \cup \sigma_m^{pf}$ and any value assignment A of σ_m^{act}, there exists exactly one stable model $I_{TC\cup A}$ of $Tr(D,m)$ that satisfies $TC \cup A$, and the probability of $I_{TC\cup A}$ is

$$Pr_{Tr(D,m)}(I_{TC\cup A}) = \frac{\prod\limits_{c=v\in TC} M(c=v)}{(|\sigma^{act}|+1)^m}.$$

The following theorem tells us that the conditional probability of transiting from a state s to another state s' with action e remains the same for all timesteps, i.e., the conditional probability of $i{+}1{:}s'$ given $i : s$ and $i : e$ correctly represents the transition probability from s to s' via e in the transition system.

Theorem 2
For any state s and s', and action e, we have

$$Pr_{Tr(D,m)}(i{+}1{:}s' \mid i : s, i : e) = Pr_{Tr(D,m)}(j{+}1{:}s' \mid j : s, j : e)$$

for any $i, j \in \{0, \dots, m-1\}$ such that $Pr_{Tr(D,m)}(i : s) > 0$ and $Pr_{Tr(D,m)}(j : s) > 0$.

For every subset X_m of $\sigma_m \setminus \sigma_m^{pf}$, let $X^i (i < m)$ be the triple consisting of

- the set consisting of atoms A such that $i : A$ belongs to X_m and $A \in \sigma^{fl}$;
- the set consisting of atoms A such that $i : A$ belongs to X_m and $A \in \sigma^{act}$;
- the set consisting of atoms A such that $i{+}1{:}A$ belongs to X_m and $A \in \sigma^{fl}$.

Let $p(X^i)$ be the transition probability of X^i, s_0 is the interpretation of σ_0^{fl} defined by X^0, and e_i be the interpretations of $i : \sigma^{act}$ defined by X^i.

Since the transition probability remains the same, the probability of a path given a sequence of actions can be computed from the probabilities of transitions.

Corollary 1
For every $m \geq 1$, X_m is a residual (probabilistic) stable model of $Tr(D,m)$ iff $X^0, \dots, X^{m-1}$ are transitions of D and $0{:}s_0$ is a residual stable model of D_{init}. Furthermore,

$$Pr_{Tr(D,m)}(X_m \mid 0{:}e_0, \dots, m-1{:}e_{m-1}) = p(X^0) \times \cdots \times p(X^m) \times Pr_{Tr(D,m)}(0{:}s_0).$$

p$\mathcal{BC}$+ with Utility

Wang and Lee (2019) has extended *p$\mathcal{BC}$+* with the notion of utility as follows.

We extend *p$\mathcal{BC}$+* by introducing the following expression called *utility law* that assigns a reward to transitions:

$$\textbf{reward}\ v\ \textbf{if}\ F\ \textbf{after}\ G \tag{A16}$$

where v is a real number representing the reward, F is a formula that contains fluent constants only, and G is a formula that contains fluent constants and action constants only (no pf, no initpf constants). We extend the signature of $Tr(D,m)$ with a set of atoms of the form (A1). We turn a utility law of the form (A16) into the LP$^{\text{MLN}}$ rule

$$\alpha : \texttt{utility}(v, i+1, id) \ \leftarrow\ (i+1 : F) \wedge (i : G) \tag{A17}$$

where id is a unique number assigned to the LP$^{\text{MLN}}$ rule and $i \in \{0, \dots, m{-}1\}$.

Given a nonnegative integer m denoting the maximum timestamp, a *p$\mathcal{BC}$+* action description

D with utility over multi-valued propositional signature σ is defined as a high-level representation of the DT-LP$^{\rm MLN}$ program $(Tr(D, m), \sigma_m^{act})$.

We extend the definition of a probabilistic transition system as follows: A *probabilistic transition system* $T(D)$ represented by a probabilistic action description D is a labeled directed graph such that the vertices are the states of D, and the edges are obtained from the transitions of D: for every transition $\langle s, e, s'\rangle$ of D, an edge labeled $e : p, u$ goes from s to s', where $p = Pr_{D_1}(1\!:\!s' \mid 0\!:\!s \wedge 0\!:\!e)$ and $u = E[U_{D_1}(0\!:\!s \wedge 0\!:\!e \wedge 1\!:\!s')]$. The number p is called the *transition probability* of $\langle s, e, s'\rangle$, denoted by $p(s, e, s')$, and the number u is called the *transition reward* of $\langle s, e, s'\rangle$, denoted by $u(s, e, s')$.

Appendix B PBCPLUS2POMDP in Compositional Way

In particular, the inputs of PBCPLUS2POMDP(COMPO) include the following:

- LP$^{\rm MLN}$ program $\Pi(m)$, parameterized with maximum timestep m, that contains LP$^{\rm MLN}$ translation of fluent dynamic laws, observation dynamic laws and utility laws with no occurrence of action constant, and static laws, as well as pf constant declarations of pf constants that occur in those causal laws (see Figure 1);
- For each group of actions $a_i \in$ in $a_1, \dots, a_n$, an LP$^{\rm MLN}$ program $\Pi_i(m) \cup C_i(m)$, parameterized with maximum timestep m; $\Pi_i(m)$ contains translation of fluent dynamic laws, observation dynamic laws and utility laws where only actions in a_i can occur in the body, as well as pf constant declarations of pf constants that occurs in those causal laws; $C_i(m)$ contains choice rules (possibly with cardinality bounds) to generate exactly one action in the group a_i; It is up to the user how to group the actions;
- Discount factor.

The system outputs the POMDP definition $M(D)$, so that $D_m = \Pi(m) \cup \Pi_1(m) \cup \dots \Pi_n(m) \cup C(m)$, where $C(m)$ is the choice rule with cardinality constraint to generate at most one action in $a_1, \dots, a_n$ for each timestep $i \in \{0, \dots, m-1\}$. The transition probabilities, observation probabilities and reward function of $M(D)$ are obtained by conjoining those from each of $\Pi \cup \Pi_i \cup C_i$ ($i \in \{1, \dots, n\}$).

Formally, let $\mathbf{S}$, Ω, $P_{M(D)}$, $O_{M(D)}$, $R_{M(D)}$ be the set of states, the set of observations, transition probabilities, observation probabilities and reward function of $M(D)$, resp. system PBCPLUS2POMDP calls LPMLN2ASP first to solve $\Pi(0)$ to obtain $\mathbf{S}$, and then $\Pi(1) \cup \Pi_i(1) \cup C_i(1)$ to obtain $P_{M(D)}$, $O_{M(D)}$, $R_{M(D)}$ as follows:

$$P_{M(D)}(s, a, s') = P_{\Pi(1)\cup\Pi_i(1)\cup C_i(1)}(1 : s' \mid 0 : s, 0 : a)$$

$$O_{M(D)}(s, a, o) = P_{\Pi(1)\cup\Pi_i(1)\cup C_i(1)}(1 : o \mid 1 : s, 0 : a)$$

$$R_{M(D)}(s, a, s') = E[U_{\Pi(1)\cup\Pi_i(1)\cup C_i(1)}(0 : s, 0 : a, 1 : s')]$$

for each $a \in a_i$, $s, s' \in \mathbf{S}$ and $o \in \Omega$.

Example 1

For the dialog example, we group the actions as follows: $\{\mathit{ConfirmItem}(i) \mid i \in \mathit{Item}\}$, $\{\mathit{ConfirmPerson}(p) \mid p \in \mathit{Person}\}$, $\{\mathit{ConfirmRoom}(r) \mid r \in \mathit{Room}\}$, $\{\mathit{WhichItem}\}$, $\{\mathit{WhichPerson}\}$, $\{\mathit{WhichRoom}\}$, $\{\mathit{Deliver}(i, p, r) \mid i \in \mathit{Item}, p \in \mathit{Person}, r \in \mathit{Room}\}$.

Π is

```
astep(0..m-1).
step(0..m).
boolean(t; f).
item(coffee; coke; cookies; burger).
person(alice; bob; carol).
room(r1; r2; r3).

% UEC
:- obs_Item(X1, I), obs_Item(X2, I), X1 != X2.
:- not obs_Item(coffee, I), not obs_Item(coke, I),
   not obs_Item(cookies, I), not obs_Item(burger, I),
   not obs_Item(na, I), step(I).
:- obs_Person(X1, I), obs_Person(X2, I), X1 != X2.
:- not obs_Person(alice, I), not obs_Person(bob, I),
   not obs_Person(carol, I), not obs_Person(na, I),
   step(I).
:- obs_Room(X1, I), obs_Room(X2, I), X1 != X2.
:- not obs_Room(r1, I), not obs_Room(r2, I),
   not obs_Room(r3, I), not obs_Room(na, I),
   step(I).
:- obs_Confirmed(X1, I), obs_Confirmed(X2, I), X1 != X2.
:- not obs_Confirmed(yes, I), not obs_Confirmed(no, I),
   not obs_Confirmed(na, I), step(I).

:- fl_ItemReq(X1, I), fl_ItemReq(X2, I), X1 != X2.
:- not fl_ItemReq(coffee, I), not fl_ItemReq(coke, I),
   not fl_ItemReq(cookies, I), not fl_ItemReq(burger, I),
   not fl_ItemReq(na, I), step(I).
:- fl_PersonReq(X1, I), fl_PersonReq(X2, I), X1 != X2.
:- not fl_PersonReq(alice, I), not fl_PersonReq(bob, I),
   not fl_PersonReq(carol, I), not fl_PersonReq(na, I),
   step(I).
:- fl_RoomReq(X1, I), fl_RoomReq(X2, I), X1 != X2.
:- not fl_RoomReq(r1, I), not fl_RoomReq(r2, I),
   not fl_RoomReq(r3, I), not fl_RoomReq(na, I),
   step(I).
:- fl_Terminated(X1, I), fl_Terminated(X2, I), X1 != X2.
:- not fl_Terminated(t, I), not fl_Terminated(f, I), step(I).

%% No two observations can occur at the same time step
:- obs_Item(It, I), obs_Person(P, I), It != na, P != na.
:- obs_Item(It, I), obs_Room(R, I), It != na, R != na.
:- obs_Item(It, I), obs_Confirmed(C, I), It != na, C != na.
:- obs_Person(P, I), obs_Room(R, I), P != na, R != na.
:- obs_Person(P, I), obs_Confirmed(C, I), P != na, C != na.
:- obs_Room(R, I), obs_Confirmed(C, I), R != na, C != na.

% Inertial Fluents
{fl_ItemReq(It, I+1)} :- fl_ItemReq(It, I), astep(I).
{fl_PersonReq(P, I+1)} :- fl_PersonReq(P, I), astep(I).
{fl_RoomReq(R, I+1)} :- fl_RoomReq(R, I), astep(I).
{fl_Terminated(B, I+1)} :- fl_Terminated(B, I), astep(I).

% Initial value of regular fluents and observation constants are exogenous
```

```
{fl_Terminated(B, 0)} :- boolean(B).
{fl_ItemReq(It, 0)} :- item(It).
{fl_PersonReq(P, 0)} :- person(P).
{fl_RoomReq(R, 0)} :- room(R).
{obs_Item(It, 0)} :- item(It).
{obs_Person(P, 0)} :- person(P).
{obs_Room(R, 0)} :- room(R).
{obs_Confirmed(yes, 0); obs_Confirmed(no, 0)}.

% By default, observation constant has na value
{obs_Item(na, I)} :- step(I).
{obs_Person(na, I)} :- step(I).
{obs_Room(na, I)} :- step(I).
{obs_Confirmed(na, I)} :- step(I).
```

Π_1 contains definition of action $Ask2ConfirmItem$:

```
% Action: ConfirmItem
:- c(It, X1, I), act_ConfirmItem(It, X2, I), X1 != X2.
:- not act_ConfirmItem(It, t, I), not act_ConfirmItem(It, f, I), item(It), astep(I).

:- pf_ConfirmWhenCorrect(X1, I), pf_ConfirmWhenCorrect(X2, I), X1 != X2.
:- not pf_ConfirmWhenCorrect(yes, I), not pf_ConfirmWhenCorrect(no, I), astep(I).
:- pf_ConfirmWhenIncorrect(X1, I), pf_ConfirmWhenIncorrect(X2, I), X1 != X2.
:- not pf_ConfirmWhenIncorrect(yes, I), not pf_ConfirmWhenIncorrect(no, I), astep(I).

@log(0.8) pf_ConfirmWhenCorrect(yes, I) :- astep(I).
@log(0.2) pf_ConfirmWhenCorrect(no, I) :- astep(I).

@log(0.2) pf_ConfirmWhenIncorrect(yes, I) :- astep(I).
@log(0.8) pf_ConfirmWhenIncorrect(no, I) :- astep(I).

obs_Confirmed(C, I+1) :- fl_ItemReq(It, I+1), fl_Terminated(f, I+1),
           act_ConfirmItem(It, t, I), pf_ConfirmWhenCorrect(C, I).
obs_Confirmed(C, I+1) :- fl_ItemReq(It, I+1), fl_Terminated(f, I+1),
           act_ConfirmItem(It1, t, I), It1 != It, pf_ConfirmWhenIncorrect(C, I).

{act_ConfirmItem(It, B, I)} :- item(It), boolean(B), astep(I).
:- not 1{act_ConfirmItem(It, t, I) : item(It)}1, astep(I).
```

Similarly, Π_2, Π_3 contains definition of actions $ConfirmPerson$ and $ConfirmRoom$.

Π_4 contains definition of actions $WhichItem(t)$:

```
% Action WhichItem
:- act_WhichItem(X1, I), act_WhichItem(X2, I), X1 != X2.
:- not act_WhichItem(t, I), not act_WhichItem(f, I), astep(I).

:- pf_WhichItem(It, X1, I), pf_WhichItem(It, X2, I), X1 != X2.
:- not pf_WhichItem(It, coffee, I), not pf_WhichItem(It, coke, I),
   not pf_WhichItem(It, cookies, I), not pf_WhichItem(It, burger, I),
   item(It), astep(I).

@log(0.7) pf_WhichItem(coffee, coffee, I) :- astep(I).
@log(0.1) pf_WhichItem(coffee, coke, I) :- astep(I).
@log(0.1) pf_WhichItem(coffee, cookies, I) :- astep(I).
```

```
@log(0.1) pf_WhichItem(coffee, burger, I) :- astep(I).
@log(0.1) pf_WhichItem(coke, coffee, I) :- astep(I).
@log(0.7) pf_WhichItem(coke, coke, I) :- astep(I).
@log(0.1) pf_WhichItem(coke, cookies, I) :- astep(I).
@log(0.1) pf_WhichItem(coke, burger, I) :- astep(I).
@log(0.1) pf_WhichItem(cookies, coffee, I) :- astep(I).
@log(0.1) pf_WhichItem(cookies, coke, I) :- astep(I).
@log(0.7) pf_WhichItem(cookies, cookies, I) :- astep(I).
@log(0.1) pf_WhichItem(cookies, burger, I) :- astep(I).
@log(0.1) pf_WhichItem(burger, coffee, I) :- astep(I).
@log(0.1) pf_WhichItem(burger, coke, I) :- astep(I).
@log(0.1) pf_WhichItem(burger, cookies, I) :- astep(I).
@log(0.7) pf_WhichItem(burger, burger, I) :- astep(I).

obs_Item(It1, I+1) :- fl_ItemReq(It, I+1), fl_Terminated(f, I+1),
           act_WhichItem(t, I), pf_WhichItem(It, It1, I).

%{act_WhichItem(B, I)} :- boolean(B), astep(I).
act_WhichItem(t, I) :- astep(I).
```

Similarly, Π_5 and Π_6 contain definitions of actions $WhichPerson(t)$ and $WhichRoom(t)$.

Π_7 contains definitions of action $Deliver(i, p, r)$:

```
% Action: Deliver
:- act_Deliver(It, P, R, X1, I), act_Deliver(It, P, R, X2, I), X1 != X2.
:- not act_Deliver(It, P, R, t, I), not act_Deliver(It, P, R, f, I), item(It), person(P), room(R), as
).

utility(1, I+1, It) :-  fl_ItemReq(It, I+1), act_Deliver(It, P, R, t, I), fl_Terminated(f, I).
utility(1, I+1, P) :-  fl_PersonReq(P, I+1), act_Deliver(It, P, R, t, I), fl_Terminated(f, I).
utility(1, I+1, R) :-  fl_RoomReq(R, I+1), act_Deliver(It, P, R, t, I), fl_Terminated(f, I).

fl_Terminated(t, I+1) :- act_Deliver(It, P, R, t, I).

{act_Deliver(It, P, R, B, I)} :- item(It), person(P), room(R), boolean(B), astep(I).
:- not 1{act_Deliver(It, P, R, t, I) : item(It), person(P), room(R)}1, astep(I).
```

Π_8 contains definitions of no-action:

```
act_noact(I) :- astep(I).
```

With this way of grouping actions, system PBCPLUS2POMDP(COMPO) can generate POMDP for this example with ~ 5 minutes.

Appendix C Tiger Example

Example 2
(**Two Tigers Example**). Consider a variant of the well-known tiger example extended with two tigers. Each of the three doors has either a tiger or a prize behind. The agent can open either of the three doors. The agent can also listen to get a better idea of where the tiger is. Listening yields the correct information about where each of the two tigers is with probability 0.85. This example can be represented in the extended $p\mathcal{BC}+$ as follows:

Notation: l, l_1, l_2, l_3 range over `Left`, `Middle`, `Right`, y ranges over `Tiger1`, `Tiger2`

Observation constants:	Domains:
TigerPositionObserved(y)	$\{\texttt{Left}, \texttt{Middle}, \texttt{Right}, \texttt{NA}\}$
Regular fluent constants:	Domains:
TigerPosition(y)	$\{\texttt{Left}, \texttt{Middle}, \texttt{Right}\}$
Action constants:	Domains:
Listen	Boolean
OpenDoor(l)	Boolean
Pf constants:	Domains:
Pf_Listen	Boolean
Pf_FailedListen(y)	$\{\texttt{Left}, \texttt{Middle}, \texttt{Right}\}$

A reward of 10 is obtained for opening the door with no tiger behind.

reward 10 **if** *TigerPosition*(`Tiger1`)$=l_1 \wedge$ *TigerPosition*(`Tiger2`)$=l_2$ **after** *OpenDoor*(l_3)
$(l_1 \neq l_3, l_2 \neq l_3)$.

A penalty of 100 is imposed on opening a door with a tiger behind.

reward -100 **if** *TigerPosition*$(y)=l$ **after** *OpenDoor*(l).

Executing the action *Listen* has a small penalty of 1.

reward -1 **if** $\top$ **after** *Listen*.

Two tigers cannot be in the same position.

caused $\perp$ **if** *TigerPosition*(`Tiger1`)$=l \wedge$ *TigerPosition*(`Tiger2`)$=l$.

Successful listening reveals the positions of the two tigers.

observed *TigerPositionObserved*$(y)=l$ **if** *TigerPosition*$(y)=l$ **after** *Listen* $\wedge$ *Pf_Listen*.

Failed listening yields a random position for each tiger.

caused *Pf_FailedListen*$(y) = \{\texttt{Left} : \frac{1}{3}, \texttt{Middke} : \frac{1}{3}, \texttt{Right} : \frac{1}{3}\}$,
observed *TigerPositionObserved*$(y)=l$ **if** $\top$ **after** *Listen*$\wedge \sim$ *Pf_Listen* $\wedge$ *Pf_FailedListen(y)*$=l$.

The positions of tigers observe the commonsense law of inertia.

inertial *TigerPosition*(y).

The action *Listen* has a success rate of 0.85.

caused *Pf_Listen* $= \{\text{TRUE} : 0.85, \text{FALSE} : 0.15\}$.