

Generalized Answer Set Planning with Incomplete Information

Javier Romero¹, Torsten Schaub^{1,2}, Tran Cao Son³

¹University of Potsdam ²INRIA Rennes ³New Mexico State University, Las Cruces

Abstract. Answer Set Planning was one of the first challenging applications of Answer Set Programming (ASP). However, when putting plans into practice, their execution and monitoring lead to an increased complexity. Foremost, this is due to the inherent incompleteness of information faced in real-world scenarios. This fundamental problem has already been addressed in various ways, as in conformant, assertional, and assumption-based planning, and often combined with additional sensing actions. Our objective is to conceive a framework for planning and execution under incomplete information by combining the key concepts of the aforementioned approaches in a uniform ASP-based framework. This paper reports on our first step in this direction. More precisely, we define a series of increasingly complex ASP problems reflecting the successive integration of the key concepts. Our prime concern lies in obtaining a uniform characterization that matches the expressive realm of ASP. This is reflected by developing appropriate approximations that are implemented by means of metaencoding techniques.

1 Introduction

Being able to operate in a dynamic environment has been one of the critical traits of intelligent agents. This includes the ability to create plans that transform the state of the world from an initial state to a desirable state. Planning is computationally expensive and it is harder when the initial state is incomplete and sensing actions are needed. Assuming that actions are deterministic and the plan's length is polynomially bounded by the size of the problem, the complexity of the planning problem in this situation is PSPACE-complete. By not considering sensing actions, the complexity of the planning problem, often referred to as conformant planning problem, is Σ_2^P -complete; using an incomplete approximation semantics lowers the complexity of the problem to NP-complete (e.g., using the 0-approximation) [6].

Planning using logic programs under the answer set semantics, or *answer set planning* (ASP), was one of the earlier applications of answer set programming [15]. It has been extended to deal with incomplete information about the initial state and/or sensing actions. More specifically, ASP has been extended to solve conformant planning [10, 18, 20] and conditional planning problems [19]. It is worth noticing that, except [10], previous approaches to planning with incomplete information and sensing actions using logic programming have to rely on the approximation semantics. The main reason lies in its low complexity. The experimental results in these works show that ASP is competitive with other approaches to planning with incompleteness and sensing actions.

ASP has been an integral part of the intelligent agent architecture *AAA* proposed in [5] and further developed in [3, 4] whose key component is the *Observe-Think-Act loop*. However, the planning problem considered in applications employing this architecture—solved during the *Think* phase of the loop—does not consider incomplete information and sensing actions.

Alternative approaches for intelligent agents to dealing with incomplete information and sensing actions include *assumption-based planning* (ABP) [9] and *continual planning* [7]. In ABP, an agent acts as though it knows the truth values of unknown fluents when it needs to create a plan; during the execution of the plan, the agent adjusts and replans when some assumptions turn out to be incorrect. Continual planning addresses the problem by providing an agent with a skeleton of a plan (or an incompletely specified plan, that can only be completed when additional information is available) that allows the agent to interleave planning and execution while guaranteeing that the intended goal is achieved. This approach is similar to that of using GOLOG [14] (or its extensions) in robotics [8]. Although there has been an attempt to extend ASP to employ GOLOG in plan generation [17], it does not deal with incomplete information and/or sensing actions; it has not been integrated into an implementation of the AAA architecture as well.

Our goal in this paper is to further investigate the use of logic programming in dealing with incomplete information in dynamic domains. Instead of focusing on the encoding of a specific planning problem (e.g., conformant, conditional, assumption-based, or continual planning), we develop an abstract framework that can eventually combine the features present in all types of planning problems. Our long term goal is to employ this framework in robotic environments and integrate it with appropriate ASP reasoning engines such as *ROSoClingo* [1].

2 Foundations

A logic program P over a set \mathcal{A} of atoms is a set of rules of form

$$a_1; \dots; a_m \leftarrow a_{m+1}, \dots, a_n, \sim a_{n+1}, \dots, \sim a_o \quad (1)$$

where each a_i is an atom in \mathcal{A} . As usual, \sim denotes (default) negation. Given a rule r as in (1), we call $\{a_1, \dots, a_m\}$ and $\{a_{m+1}, \dots, a_n, \sim a_{n+1}, \dots, \sim a_o\}$ its *head* and *body* and denote them by $H(r)$ and $B(r)$. Moreover, we define $B(r)^+ = \{a_{m+1}, \dots, a_n\}$ and $B(r)^- = \{a_{n+1}, \dots, a_o\}$. A rule is called *fact* if $m, o = 1$, *normal* if $m = 1$, and an *integrity constraint* if $m = 0$. Semantically, logic programs induce sets of stable models, which are distinguished models of the program determined by stable models semantics; see [13] for details.

To ease the use of ASP in practice, several extensions have been developed. First of all, rules with variables are viewed as shorthands for the set of their ground instances. An integrity constraint r abbreviates the rule $x \leftarrow B(r), \sim x$ where x is a new symbol. Similarly, for $a \in \mathcal{A}$, a choice rule r is of form $\{a\} \leftarrow B(r)$ and stands for the pair of rules $a \leftarrow B(r), \sim a'$ and $a' \leftarrow B(r), \sim a$ where a' is a new symbol associated with a . We define $H(r) = \{a\}$ for a choice rule as r . Further language constructs include conditional literals and cardinality constraints [16]. The former are of the form

$a : b_1, \dots, b_m$, the latter can be written as $s \{d_1; \dots; d_n\} t$, where a and b_i are possibly default-negated (regular) literals and each d_j is a conditional literal; s and t provide optional lower and upper bounds on the number of satisfied literals in the cardinality constraint. We refer to b_1, \dots, b_m as a condition. The practical value of both constructs becomes apparent when used with variables. For instance, a conditional literal like $a(X) : b(X)$ in a rule's antecedent expands to the conjunction of all instances of $a(X)$ for which the corresponding instance of $b(X)$ holds. Similarly, $2 \{a(X) : b(X)\} 4$ is true whenever at least two and at most four instances of $a(X)$ (subject to $b(X)$) are true. Finally, assignments are of form $t = \{d_1; \dots; d_n\}$. E.g., $Y = \{a(X) : b(X)\}$ is true if Y equals the number of satisfied instances of $a(X) : b(X)$.

We identify the choice rules, normal rules, and integrity constraint in a logic program P by means of $c(P)$, $n(P)$, and $i(P)$, respectively. We define a program P as *normalized*, if it can be partitioned into $c(P)$, $n(P)$, and $i(P)$, such that $B(r) = \emptyset$ for each $r \in c(P)$ and $n(P)$ is stratified [2]. For simplicity, in this paper we restrict ourselves to normalized programs, although the implementation handles arbitrary programs.

In what follows, we rely on reification for translating a logic program P into a set of facts

$$R(P) = \{choice(a) \leftarrow \mid r \in c(P), a \in H(r)\} \quad (2)$$

$$\cup \{rule(r) \leftarrow \mid r \in n(P) \cup i(P)\} \quad (3)$$

$$\cup \{head(r, a) \leftarrow \mid r \in n(P), a \in H(r)\} \quad (4)$$

$$\cup \{body(r, +, a) \leftarrow \mid r \in n(P) \cup i(P), a \in B(r)^+\} \quad (5)$$

$$\cup \{body(r, -, a) \leftarrow \mid r \in n(P) \cup i(P), a \in B(r)^-\} \quad (6)$$

$$\cup \{scc(c, a) \leftarrow \mid a \in C_c, C_c \in SCC(P)\} \quad (7)$$

where $SCC(P)$ contains all strongly connected components of the positive atom dependency graph of P .¹ As an example, consider the program P_1 consisting of the rules

$$\begin{aligned} \{a\} &\leftarrow \\ q &\leftarrow a, \sim b \\ &\leftarrow \sim q \end{aligned}$$

along with its reification $R(P_1)$:

$$\begin{aligned} choice(a) &\leftarrow, \\ rule(1) &\leftarrow, \quad head(1, q) \leftarrow, \quad body(1, +, a) \leftarrow, \quad body(2, -, b) \leftarrow, \\ rule(2) &\leftarrow, \quad body(2, -, q) \leftarrow \end{aligned}$$

We interpret such reified programs by means of the meta encoding M consisting of the rules in (8) to (24); cf. [12]:

$$1\{t(X); f(X)\}1 \leftarrow choice(X) \quad (8)$$

$$tbody(R) \leftarrow rule(R), t(X) : body(R, +, X); f(Y) : body(R, -, Y) \quad (9)$$

$$fbody(R) \leftarrow rule(R), f(X), body(R, +, X) \quad (10)$$

$$fbody(R) \leftarrow rule(R), t(Y), body(R, -, Y) \quad (11)$$

¹ This information is also obtained by *gringo*'s reified output.

$$t(X) \leftarrow \text{head}(R, X), \text{tbody}(R) \quad (12)$$

$$f(X) \leftarrow \text{atom}(X), \sim \text{ab}(X), \text{fbody}(R) : \text{head}(R, X) \quad (13)$$

$$\text{internal}(C, R) \leftarrow \text{scc}(C, X), \text{head}(R, X), \text{scc}(C, X'), \text{body}(R, +, X') \quad (14)$$

$$\text{wait}(C, X, 0) \leftarrow \text{scc}(C, X), \sim \text{ab}(X), \text{fbody}(R) : \text{head}(R, X), \sim \text{internal}(C, R) \quad (15)$$

$$\text{wait}(C, R, I) \leftarrow \text{internal}(C, R), \text{body}(R, +, X), \text{wait}(C, X, I) \quad (16)$$

$$\text{wait}(C, X, I) \leftarrow \text{wait}(C, X, 0), Z = \{\text{scc}(C, -)\}, I = 1 .. Z, \quad (17)$$

$$\text{wait}(C, R, I - 1) : \text{head}(R, X), \text{internal}(C, R) \quad (18)$$

$$f(X) \leftarrow \text{scc}(C, X), \text{wait}(C, X, Z), Z = \{\text{scc}(C, -)\} \quad (19)$$

$$\text{normal}(R) \leftarrow \text{head}(R, X) \quad (20)$$

$$\leftarrow \text{rule}(R), \sim \text{normal}(R), \sim \text{fbody}(R) \quad (21)$$

$$\text{atom}(X) \leftarrow \text{head}(R, X) \quad (22)$$

$$\text{atom}(X) \leftarrow \text{body}(R, V, X) \quad (23)$$

$$\text{ab}(X) \leftarrow \text{choice}(X) \quad (24)$$

Note that the rules in Lines (9) to (13) amount to Fitting's operator [11]. This is extended by computing unfounded atoms to the well-founded operator [21] in Lines (14) to (19). Finally, the rules in Lines (20) and (21) account for integrity constraints.

Proposition 1. *The stable models of a normalized logic program P correspond one-to-one to the stable models of $R(P) \cup M$.*

For instance, the unique stable model of P_1 is $\{a, q\}$; it corresponds to the only stable model of $R(P_1) \cup M$ containing $t(a)$, $f(b)$, and $t(q)$. Accordingly, neither $P_1 \cup \{b \leftarrow\}$ nor $R(P_1) \cup \{\text{rule}(\beta) \leftarrow, \text{head}(\beta, b) \leftarrow\} \cup M$ have a stable model.

Our goal in this paper is to develop an abstract framework for planning with incomplete information that builds on a basic ASP encoding. As such, we present this encoding through an example.

Example 1. Consider a cleaning robot in a house with two rooms 1 and 2 connected through a door. Initially, the robot is at 1, the door is open, and both rooms are not clean. The robot can move between rooms if the door is open. It can also sweep the room it is in. The goal is to clean both rooms. Obviously, the robot can start with sweeping room 1, then going to room 2, and sweeping room 1.

A logic program encoding this problem would use atoms of the form $\text{holds}(f, t)$ and $\text{occ}(a, t)$ where f is a fluent in $\{\text{clean}(1), \text{clean}(2), \text{at}(1), \text{at}(2)\}$, a is an action in $\{\text{move}(1, 2), \text{move}(2, 1), \text{sweep}(1), \text{sweep}(2)\}$, t is an integer in the range $0.. \text{max}$, and max is an integer denoting the maximal desirable plan length. The basic ASP encoding contains rules for

- expressing the preconditions of actions, e.g., to state that robot can only sweep a room R if it is in the room R :

$$\text{possible}(\text{sweep}(R), T) \leftarrow \text{holds}(\text{at}(R), T)$$

- expressing the direct effects of actions, e.g., to state that if the robot sweeps room R at step T then room R is clean at step $T + 1$:

$$\text{holds}(\text{clean}(R), T + 1) \leftarrow \text{occ}(\text{sweep}(R), T)$$

- expressing the indirect effects of actions, e.g., to state that the robot is not at one room if it is at another room:

$$\text{holds}(\text{neg}(\text{at}(R)), T) \leftarrow \text{holds}(\text{at}(R'), T), R \neq R'$$

The program also contains rules for expressing inertia, rules for generating action occurrences, rules for goal verification, and rules encoding the initial states. For this problem, the initial state is encoded by

$$\text{holds}(\text{at}(1), 0) \leftarrow \tag{25}$$

$$\text{holds}(\text{neg}(\text{clean}(1)), 0) \leftarrow \text{holds}(\text{neg}(\text{clean}(2)), 0) \leftarrow \tag{26}$$

For $\text{max} = 3$, the program returns the plan $[\text{sweep}(1); \text{move}(1, 2); \text{sweep}(2)]$.

3 $\exists\forall$ ASP

In this section, we introduce the notion of a $\exists\forall$ ASP problem, similar to that of a quantified Boolean formula in propositional logic, to capture conformant planning problems. More specifically, a conformant planning problem is viewed as a program with two sets of atoms, one characterizes the set of possible plans, and the other the incomplete initial state. This notion allows us to deal with incompleteness keeping intact the ASP encoding of the planning problem except for the rules representing the initial state. For example, assume that the robot in Example 1 does not know whether any room is clean, we only need to declare that the problem is concerned with the situation where $\text{clean}(1)$ and $\text{clean}(2)$ are initially unknown. In other words, the notion of $\exists\forall$ ASP problem provides an elaboration tolerant way for dealing with incompleteness that is different from previous works such as [18, 20] or develops a specific language for this purpose as in [10].

Definition 1 ($\exists\forall$ ASP Problem). An $\exists\forall$ ASP problem is a tuple $(P, \mathcal{A}, X, Y, Z)$ where P is a logic program over \mathcal{A} , and (X, Y, Z) is a partition of \mathcal{A} .

For simplicity, we sometimes use $\exists X\forall Y P$ to refer to the $\exists\forall$ ASP problem $(P, \mathcal{A}, X, Y, Z)$, leaving implicit the sets \mathcal{A} of atoms appearing in P and $Z = \mathcal{A} - (X \cup Y)$. Let P_r be the program in Example 1 without the rules in (26) encoding part of the initial state, \mathcal{A} be its set of atoms, $X = \{\text{occ}(a, t) \mid a \text{ is an action, } t \text{ is a step}\}$, and $Y = \{\text{holds}(\text{clean}(1), 0), \text{holds}(\text{clean}(2), 0)\}$, then $(P_r, \mathcal{A}, X, Y, Z)$, also written $\exists X\forall Y P_r$, represents the conformant planning problem for the cleaning robot in Example 1.

We sometimes abuse notation and identify a set X of atoms with the set of facts $\{a \leftarrow \mid a \in X\}$. Next we define the solutions to $\exists\forall$ ASP problems.

Definition 2 ($\exists\forall$ ASP Solution). A set of atoms $X' \subseteq X$ is a solution to the $\exists\forall$ ASP problem $(P, \mathcal{A}, X, Y, Z)$ if for all sets $Y' \subseteq Y$, the program $X' \cup Y' \cup P$ has some stable model.

There is a unique solution to the problem $\exists X\forall Y P_r$ of Example 1. It contains the atoms $\text{occ}(\text{sweep}(1), 0)$, $\text{occ}(\text{move}(1, 2), 1)$, and $\text{occ}(\text{sweep}(2), 2)$. In the following, we illustrate our approach with simple examples, where letters a, b, \dots stand for actions, letter $f, g \dots$ stand for properties of the domain, u is another property which is typically unknown, and q is some query that has to be achieved.

Example 2. Let P_2 be the logic program

$$\begin{array}{ll} f \leftarrow u & \leftarrow \sim q \\ g \leftarrow a, f & q \leftarrow g \\ h \leftarrow a, \sim f & q \leftarrow h \end{array}$$

The solutions to the $\exists\{a, u\}\forall\{\}\{P_2\}$ problem are $\{a\}$ and $\{a, u\}$, and the unique solution to the $\exists\{a\}\forall\{u\}\{P_2\}$ problem is $\{a\}$.

Example 3. Let P_3 be the result of extending P_2 with the set of rules

$$h \leftarrow b \qquad \leftarrow a, b$$

The solutions to $\exists\{a, b\}\forall\{u\}\{P_3\}$ are both $\{a\}$ and $\{b\}$.

We compute the solutions to $\exists X\forall Y P$ with an encoding extending $R(P) \cup M$. For simplicity, we assume that atoms in X and Y do not appear in any rule heads in P , i.e., $(X \cup Y) \cap \{a \in H(r) \mid r \in c(P) \cup n(P)\} = \emptyset$, although this restriction may easily be lifted. Let $F(X, Y)$ be the set of facts

$$\{exists(a) \leftarrow \mid a \in X\} \cup \{forall(a) \leftarrow \mid a \in Y\}$$

reifying the atoms in X and Y , and let EF be the set of rules

$$choice(X) \leftarrow exists(X) \tag{27}$$

$$ab(X) \leftarrow forall(X) \tag{28}$$

The rule in (27) allows for choosing the truth values of the existentially quantified atoms. Without the one in (28) we would derive $f(a)$ for all atoms a in Y , since we are assuming that atoms in Y are excluded from rule heads. Observe that now in the meta encoding M no rule can infer $t(a)$ nor $f(a)$ for any a in Y . Hence, every atom derived using the meta encoding is independent of the truth value of the atoms $a \in Y$. Then, the rules in Lines (20) and (21) guarantee that the integrity constraints of the original program are not violated for any truth value of the atoms $a \in Y$. For normalized programs, this implies that the choices made on atoms from X always lead to a stable model, for every possible $Y' \subseteq Y$.

In the following, we write $E(\exists X\forall Y P)$ to denote the encoding $R(P) \cup M \cup F(X, Y) \cup EF$. The next theorem establishes that it can be used to compute solutions of $\exists X\forall Y P$.

Theorem 1. *Let $(P, \mathcal{A}, X, Y, Z)$ be an $\exists\forall ASP$ problem, then for every stable model S of the logic program $E(\exists X\forall Y P)$, $\{a \mid t(a) \in S\}_X$ is a solution to the problem.*

Example 4. (Cont. Example 2). The encoding $E(\exists\{a, u\}\forall\{\}\{P_2\})$ has two stable models, one contains atoms $t(a), f(u), f(f), f(g), t(h), t(q)$, and the other $t(a), t(u), t(f), t(g), f(h), t(q)$. They correspond to the solutions $\{a\}$ and $\{a, u\}$.

The next examples show that in general the encoding $E(\exists X\forall Y P)$ is not complete, i.e., there might be solutions it cannot compute.

Example 5. (Cont. Example 2). The problem $\exists\{a\}\forall\{u\}P_2$ has one solution $\{a\}$, but $E(\exists\{a\}\forall\{u\}P_2)$ has no stable model. This example shows that our encoding cannot reason by cases, which is needed to find the solution: if u is true, using a we obtain q after deriving f and g , and if u is false, we obtain q after deriving h , so in every case we could obtain q .

Example 6. (Cont. Example 3). The encoding $E(\exists\{a, b\}\forall\{u\}P_3)$ has one stable model with $f(a), t(b), f(g), t(h), t(q)$, corresponding to the solution $\{b\}$ of $\exists\{a, b\}\forall\{u\}P_3$, but there is no stable model for the solution $\{a\}$.

All our encodings extend $E(\exists X\forall Y P)$ and are therefore incomplete. For this reason we say that they are *approximations* to the problems they aim to solve.

3.1 Extended Solutions to $\exists\forall$ ASP

The stable models of the encoding $E(\exists X\forall Y P)$ may contain atoms $t(a)$ or $f(a)$ for some $a \in Z$. To capture their meaning we define the extended solutions to a $\exists\forall$ ASP problem, which may assign atoms $a \in Z$ to true or false. To this end, we need some extra notation. Let $X, Y \subseteq \mathcal{A}$ be sets of atoms. By X_Y we denote the set $X \cap Y$. A partial interpretation over X is a pair $\langle T, F \rangle$ where $T, F \subseteq X$ and $T \cap F = \emptyset$. We say that $\langle T, F \rangle$ is total on X if $T_X \cup F_X = X$. We define a preorder \succeq over partial interpretations, where $\langle T, F \rangle$ is more specific than $\langle T', F' \rangle$, written $\langle T, F \rangle \succeq \langle T', F' \rangle$, if $T \supseteq T'$, and $F \supseteq F'$.

Definition 3 ($\exists\forall$ ASP Extended Solution). *A partial interpretation $\langle T, F \rangle$ over $X \cup Z$ being total on X is an extended solution to the $\exists\forall$ ASP problem $(P, \mathcal{A}, X, Y, Z)$ if for all sets $Y' \subseteq Y$, the program $T_X \cup Y' \cup P \cup \{ \leftarrow \sim a \mid a \in T_Z \} \cup \{ \leftarrow a \mid a \in F_Z \}$ has some stable model.*

It turns out that, considering the atoms $t(a)$ and $f(a)$ for $a \in X \cup Z$, every stable model of $E(\exists X\forall Y P)$ corresponds to an extended solution to the problem. Note that in Theorem 1 we considered only the $t(a)$ for $a \in X$.

Theorem 2. *Let $(P, \mathcal{A}, X, Y, Z)$ be an $\exists\forall$ ASP problem, then for every stable model S of the logic program $E(\exists X\forall Y P)$, $\langle \{a \mid t(a) \in S\}_{X \cup Z}, \{a \mid f(a) \in S\}_{X \cup Z} \rangle$ is an extended solution to the problem.*

The following propositions establish the relation between solutions and extended solutions to an $\exists\forall$ ASP problem.

Proposition 2. *Let X' be a solution to the $\exists\forall$ ASP problem $(P, \mathcal{A}, X, Y, Z)$, then $\langle X', X - X' \rangle$ is an extended solution to the problem.*

Proposition 3. *Let $\langle T, F \rangle$ be an extended solution to the $\exists\forall$ ASP problem $(P, \mathcal{A}, X, Y, Z)$, then T_X is a solution to the problem.*

We say that an extended solution $\langle T, F \rangle$ is *maximal*, if it is maximal among all extended solutions wrt the ordering \succeq . It is easy to see that if $\langle T, F \rangle$ is a maximal extended solution, then all partial interpretations $\langle T', F' \rangle$ with $\langle T'_X, F'_X \rangle = \langle T_X, F_X \rangle$ and $\langle T'_Z, F'_Z \rangle \preceq \langle T_Z, F_Z \rangle$ are also extended solutions. In particular, $\langle T_X, F_X \rangle$ is an extended solution.

Example 7. (Cont. Example 4). The $\exists\forall$ ASP $\exists\{a, u\}\forall\{ \}P_2$ problem has two maximal extended solutions, $\langle\{a, h, q\}, \{u, f, g\}\rangle$, and $\langle\{a, u, f, g, q\}, \{h\}\rangle$, which correspond to the two stable models of $E(\exists\{a, u\}\forall\{ \}P_2)$ described in Example 4. Some non maximal extended solutions are, for example, $\langle\{a\}, \{u\}\rangle$ and $\langle\{a, u, q\}, \{ \}\rangle$.

Example 8. (Cont. Example 5). The extended solutions to $\exists\{a\}\forall\{u\}P_2$ are $\langle\{a, q\}, \{ \}\rangle$ and $\langle\{a\}, \{ \}\rangle$; only the first one is maximal. None of them is computed by $E(\exists\{a\}\forall\{u\}P_2)$, that has no stable models.

Example 9. (Cont. Example 6). The maximal extended solutions to $\exists\{a, b\}\forall\{u\}P_3$ are $\langle\{a, q\}, \{b\}\rangle$ and $\langle\{b, h, q\}, \{a, g\}\rangle$. The last one corresponds to the unique stable model of the encoding $E(\exists\{a, b\}\forall\{u\}P_3)$, described in Example 6.

In general, not all stable models of $E(\exists X\forall Y P)$ correspond to maximal extended solutions. For example, deleting the constraint $\leftarrow \sim q$ from P_2 , the unique stable model of $E(\exists\{a\}\forall\{u\}P_2)$ would contain $t(a)$ and correspond to the extended solution $\langle\{a\}, \{ \}\rangle$, which is less specific than the maximal extended solution $\langle\{a, q\}, \{ \}\rangle$.

4 Programs with Knowledge

$\exists\forall$ ASP is sufficiently expressive for representing conformant planning problems, but it cannot be used to represent problems that require sensing actions, i.e., reasoning about beliefs/knowledge. For instance, consider a variation of the problem in Example 1. Assume that initially the door is closed. The door may either have a key lock or a card lock. The robot has a key and a card, but the type of the lock is initially unknown. If the robot tries to open the door with the wrong object, the lock gets stucked so that afterwards there is no way to open the door. The robot can *sense* the type of the lock with a sensing action *check*. It is easy to see that this problem has no conformant plan, while there is a conditional plan with two branches after the execution of the sensing action *check*. Another approach for solving this problem is to introduce an assertion action *open* [7] establishing that the door is open if the type of the lock is known. Then there is a sequential plan that contains actions *check* and *open*, and achieves the goal. Assertions are abstract actions that cannot be executed in the real world, so after sensing the type of the lock, the robot must plan again. Observe that in this situation the type of the lock is already known, so there is again a sequential plan achieving the goal, using the appropriate object to open the door.

To represent such an approach, we introduce the notion of a logic program with knowledge and extend the notion of $\exists\forall$ ASP to this new class of programs. If X is a set of atoms, by X^{kw} we denote the set $\{a^{kw} \mid a \in X\}$ of know-whether atoms of X .

Definition 4 (Logic program with knowledge). A logic program with knowledge P over \mathcal{A} is a logic program over $\mathcal{A} \cup \mathcal{A}^{kw}$ where $\mathcal{A}^{kw} \cap \{a \mid r \in P, a \in H(r)\} = \emptyset$.

The last condition expresses that the know-whether atoms \mathcal{A}^{kw} are not allowed to appear in the head of the rules. Given a partial interpretation $\langle T, F \rangle$ over \mathcal{A} , we say that an atom $a \in \mathcal{A}$ is (*known to be*) *true* if $a \in T$, it is (*known to be*) *false* if $a \in F$, it is

known(-whether) if $a \in T \cup F$, and otherwise it is *unknown(-whether)*. The know-whether atoms of logic programs with knowledge allow expressing that an atom a is known or unknown in a *partial interpretation*, via a^{kw} or $\sim a^{kw}$, respectively. They are restricted to the bodies of rules to guarantee that their truth value is determined by the partial interpretation they are referring to. We next define the notions of a problem with knowledge and the corresponding solution.

Definition 5 ($\exists\forall$ ASP Problem with Knowledge). A $\exists\forall$ ASP problem with knowledge is a tuple $(P, \mathcal{A}, X, Y, Z)$ where P is a logic program with knowledge over \mathcal{A} , and (X, Y, Z) is a partition of \mathcal{A} .

Definition 6 ($\exists\forall$ ASP Solution with Knowledge). A partial interpretation $\langle T, F \rangle$ over $X \cup Z$ being total on X is a solution to the $\exists\forall$ ASP problem with knowledge $(P, \mathcal{A}, X, Y, Z)$ if for all sets $Y' \subseteq Y$, the program $T_X \cup Y' \cup P \cup \{ \leftarrow \sim a \mid a \in T_Z \} \cup \{ \leftarrow a \mid a \in F_Z \} \cup \{ a^{kw} \leftarrow a \mid a \in T \} \cup \{ a^{kw} \leftarrow \sim a \mid a \in F \}$ has some stable model.

As before, we say that a solution to a $\exists\forall$ ASP problem with knowledge is *maximal* if it is a maximal element of the set of solutions to the problem ordered by \succeq .

Example 10. The problem with knowledge $\exists\{\} \forall\{u\} P_4$ where P_4 is the following logic program

$$\begin{array}{lll} f \leftarrow & \leftarrow g & h \leftarrow u \\ i \leftarrow f^{kw}, g^{kw} & j \leftarrow \sim g^{kw} & k \leftarrow \sim h^{kw} \end{array}$$

has a unique maximal solution $\langle \{f, i, k\}, \{g, j\} \rangle$. From the rules in the first line we obtain that f is known to be true, g is known to be false, and h is unknown. From the second line, given that f and g are known, we derive i but not j , and we obtain k since h is unknown.

Given a partial interpretation $\langle T, F \rangle$, the rules $\{ a^{kw} \leftarrow a \mid a \in T \} \cup \{ a^{kw} \leftarrow \sim a \mid a \in F \}$ of Definition 6 infer the atoms a^{kw} for every $a \in T \cup F$. In the Example 10, they derive atoms f^{kw} and g^{kw} , which are used to obtain i and block the derivation of j . Note that simply adding the set of rules $\{ a^{kw} \leftarrow a \mid a \in \mathcal{A} \} \cup \{ a^{kw} \leftarrow \sim a \mid a \in \mathcal{A} \}$ to Definition 6 would not work, because then a^{kw} would always be true for all $a \in \mathcal{A}$.

The know-whether atoms a^{kw} express that either a is known to be true or a is known to be false in a partial interpretation. We can introduce new atoms a^{kt} and a^{kf} for expressing that a is known true in a partial interpretation, and that a is known false in a partial interpretation, respectively. They can be defined with the rules

$$\{ a^{kt} \leftarrow a, a^{kw} \mid a \in \mathcal{A} \} \cup \{ a^{kf} \leftarrow \sim a, a^{kw} \mid a \in \mathcal{A} \} \quad (29)$$

Example 11. (Cont. Example 10) The problem $\exists\{\} \forall\{u\} P_4 \cup \{ \leftarrow h \}$ has no solution, since we cannot prove that h is false for all values of u . However, adding (29) and replacing $\leftarrow h$ by $\leftarrow h^{kt}$, the problem $\exists\{\} \forall\{u\} P_4 \cup (29) \cup \{ \leftarrow h^{kt} \}$ has some solution again, since now the constraint only requires h not *known* to be true.

The next theorem states the relation between the solutions of a problem with knowledge and the solutions defined in the previous section.

Theorem 3. A partial interpretation $\langle T, F \rangle$ over $X \cup Z$ being total on X is a solution to the $\exists\forall$ ASP problem with knowledge $(P, \mathcal{A}, X, Y, Z)$ if $\langle T, F \rangle$ is an extended solution to the $\exists\forall$ ASP problem (without knowledge) $(P', \mathcal{A} \cup \mathcal{A}^{kw}, X, Y, Z \cup \mathcal{A}^{kw})$ where $P' = P \cup \{a^{kw} \leftarrow a \mid a \in T\} \cup \{a^{kw} \leftarrow \sim a \mid a \in F\}$.

The theorem reduces a problem with knowledge to a problem without it, where the know-whether atoms are treated as regular atoms, and the original program is extended with the rules to derive them appropriately.

Example 12. The problem with knowledge $\exists\{\}\forall\{u\}P_5$ where P_5 is the following logic program

$$\begin{array}{ll} f \leftarrow u & \textit{sense} \leftarrow \sim f^{kw} \\ f \leftarrow \sim u & \end{array}$$

has two maximal solutions: $\langle \{f\}, \{\textit{sense}\} \rangle$ and $\langle \{\textit{sense}\}, \{\}\rangle$. In the first solution the atom f is true and \textit{sense} cannot be derived, while in the second solution f is unknown and \textit{sense} can be obtained.

Usually, approaches to knowledge consider that an atom is known if it holds for all possible values of the unknown atoms. Under this interpretation, in the previous example, only $\langle \{f\}, \{\textit{sense}\} \rangle$ would be a valid solution, since no matter the value of u , atom f is always true, so we can conclude that it is known and therefore \textit{sense} is false. However, for an agent using an approximation, the solution $\langle \{\textit{sense}\}, \{\}\rangle$ is also reasonable. The agent using our encoding actually does not know the value of f , given that it cannot reason by cases. Then it is reasonable for the agent to derive \textit{sense} , so that afterwards it senses the value of f . In this respect, our approach to knowledge does not capture the notion of *what may be known* by an agent using a logic program, but rather it represents *what is actually known* by that agent using an approximation.

Now let's consider an example with an assertion action.

Example 13. Let P_6 be the program that results after replacing in P_2 (from Example 2) the rule

$$h \leftarrow a, \sim f$$

by these two

$$h \leftarrow b, \sim f \qquad \leftarrow a, b$$

The problem $\exists\{a, b\}\forall\{u\}P_6$ has no solution. Intuitively, there is a conditional solution where if u is true, then a is made true to achieve q , while if u is false, then b is made true instead. This simple type of conditional solution can be modeled with an assertion. Let P_7 extend P_6 with the following rules

$$f' \leftarrow f^{kw} \qquad f' \leftarrow \textit{sense} \qquad q \leftarrow c, f'$$

where f' is a new atom representing that f is known either because we know-whether it or because we have sensed it, \textit{sense} senses fluent f , and c stands for an assertion that derives q when f' is true. The problem $\exists\{c, \textit{sense}\}\forall\{u\}P_7$ has one maximal solution $\langle \{c, \textit{sense}, f', q\}, \{a, b, g, h\} \rangle$. Atoms f and f' are unknown and true, respectively,

representing the case where although f is not known-whether, it is known because it has been sensed. In the real-world, the agent would first sense f , and afterwards, given that the assertion c is an abstract action that cannot be executed, the agent would have to search for a new solution using the knowledge gathered by sensing.

For computing the solutions to a $\exists\forall$ ASP problem with knowledge $(P, \mathcal{A}, X, Y, Z)$, we extend the encoding $E(\exists X\forall Y P)$ with the set of rules K

$$\begin{aligned} & \{t(a^{kw}) \leftarrow t(a) \mid a \in \mathcal{A}\} \cup \{t(a^{kw}) \leftarrow f(a) \mid a \in \mathcal{A}\} \cup \\ & \{ab(a^{kw}) \leftarrow t(a) \mid a \in \mathcal{A}\} \cup \{ab(a^{kw}) \leftarrow f(a) \mid a \in \mathcal{A}\} \end{aligned}$$

adding appropriately the rules $\{a^{kw} \leftarrow a \mid a \in T\} \cup \{a^{kw} \leftarrow \sim a \mid a \in F\}$ of Definition 6. When an atom a is known true or false in a partial interpretation $\langle T, F \rangle$, the encoding $E(\exists X\forall Y P)$ derives $t(a)$ or $f(a)$, respectively. Then the rules in the first line infer $t(a^{kw})$, and the rules in the second line block the establishment of $f(a^{kw})$ by the meta encoding. On the other hand, if a is unknown in $\langle T, F \rangle$, then the rules in K are not applied, while the rules in the meta encoding infer $f(a^{kw})$, given that know-whether atoms do not appear in the head of rules.

Theorem 4. *Let $(P, \mathcal{A}, X, Y, Z)$ be an $\exists\forall$ ASP problem with knowledge, then for each stable model S of the logic program $E(\exists X\forall Y P) \cup K$, $\langle \{a \mid t(a) \in S\}_{X \cup Z}, \{a \mid f(a) \in S\}_{X \cup Z} \rangle$ is a solution to the problem.*

Example 14. (Cont. Example 10). The encoding $E(\exists\{\} \forall\{u\} P_4) \cup K$ has one stable model, corresponding to the unique maximal solution to the problem.

Example 15. (Cont. Example 11). The encoding $E(\exists\{\} \forall\{u\} P_4 \cup \{\leftarrow h\}) \cup K$ has no stable model, while $E(\exists\{\} \forall\{u\} P_4 \cup (29) \cup \{\leftarrow h^{kt}\}) \cup K$ has one, corresponding to the unique maximal solution to the problem.

Example 16. (Cont. Example 12). The encoding $E(\exists\{\} \forall\{u\} P_5) \cup K$ has only one stable model, corresponding to the maximal solution $\langle \{sense\}, \{\} \rangle$ to the problem. There is no stable model for the solution $\langle \{f\}, \{sense\} \rangle$, given that it would require reasoning by cases.

Example 17. (Cont. Example 13). The encoding $E(\exists\{a, b\} \forall\{u\} P_6) \cup K$ has no stable model, while $E(\exists\{c, sense\} \forall\{u\} P_7) \cup K$ has one stable model corresponding to the unique maximal solution to the problem.

5 Solutions with Assumptions

In general, for dealing with incomplete information and sensing actions, agents may have to devise conditional plans, or plans with assertions. However, as argued in [9], this might be not necessary in many practical situations, where agents can still achieve their goals by making *reasonable assumptions*, planning, and executing the generated plan with the understanding that replanning might be needed sometimes. Let us reconsider the variation of the problem in Example 1 discussed in Section 4 with a slight change: when the robot tries to open the door with the wrong object, the door does not open,

and there is a signal indicating the error. In this domain, the robot could proceed by *assuming* that the key can be used to open the door, and generating a plan involving this assumption (e.g., a reasonable plan is the sequence: sweep room 1, open the door with the key, move to b , and sweep room 2). The assumption is justified because the robot is familiar with this type of arrangements, and it has encountered far more doors with the key lock than with the card lock. When the robot executes the plan, if it realizes that the assumption was incorrect, it can address the issue by replanning. To represent this approach, we introduce the notion of a solution with assumptions, extending the notion of solution of a $\exists\forall$ ASP problem.

Definition 7 (Solution with Assumptions of $\exists\forall$ ASP). A partial interpretation $\langle T, F \rangle$ over \mathcal{A} being total on X is a solution with assumptions $\langle T_Y, F_Y \rangle$ to the $\exists\forall$ ASP problem $(P, \mathcal{A}, X, Y, Z)$ if $\langle T, F \rangle$ is a solution to the $\exists\forall$ ASP problem $(P \cup T_Y \cup \{\leftarrow a \mid a \in F_Y\}, \mathcal{A}, X, Y - (T_Y \cup F_Y), Z \cup T_Y \cup F_Y)$.

A solution with assumptions assumes truth values for a subset of the universally quantified atoms, and reduces the problem to one without assumptions. This is achieved extending the original program with facts for the atoms that are assumed to be true, and constraints for the atoms that are assumed to be false. Furthermore, the selected atoms are no longer quantified.

The implementation only requires allowing choices over the universally quantified atoms. This can be done with the following program A :

$$\{choice(X)\} \leftarrow forall(X)$$

Theorem 5. Let $(P, \mathcal{A}, X, Y, Z)$ be an $\exists\forall$ ASP problem, then for each stable model S of the logic program $E(\exists X\forall Y P) \cup A$, $\langle \{a \mid t(a) \in S\}_{\mathcal{A}}, \{a \mid f(a) \in S\}_{\mathcal{A}} \rangle$ is a solution with assumptions $\langle \{a \mid t(a) \in S\}_Y, \{a \mid f(a) \in S\}_Y \rangle$ to the problem.

Example 18. (Cont. Example 2). Consider the problem $\exists\{a\}\forall\{u\}P_2$ of Example 2. As shown in Example 5, the solution $\langle \{a, q\}, \{\} \rangle$ cannot be computed by the program $E(\exists\{a\}\forall\{u\}P_2)$, but using $E(\exists\{a\}\forall\{u\}P_2) \cup A$ two solutions with assumptions can be computed. The first is $\langle \{a, f, g, q, u\}, \{h\} \rangle$ with assumptions $\langle \{u\}, \{\} \rangle$, and the second is $\langle \{a, h, q\}, \{f, g, u\} \rangle$ with assumptions $\langle \{\}, \{u\} \rangle$.

Example 19. (Cont. Example 13). Consider the problem $\exists\{a, b\}\forall\{u\}P_6$ of Example 13 with no solutions. There are two solutions with assumptions that can be computed with $E(\exists\{a, b\}\forall\{u\}P_6) \cup A$. The first is $\langle \{a, f, g, q, u\}, \{b, h\} \rangle$ with assumptions $\langle \{u\}, \{\} \rangle$, and the second is $\langle \{b, h, q\}, \{a, f, g, u\} \rangle$ with assumptions $\langle \{\}, \{u\} \rangle$.

Among the solutions with assumptions computed by $E(\exists X\forall Y P) \cup A$, we can select those with a minimal number of assumptions simply adding a minimize statement:

$$\#minimize\{1, X : choice(X), forall(X)\}$$

Finally, the previous notions are seamlessly applied to logic programs with knowledge.

Definition 8 (Solution with Assumptions to $\exists\forall$ ASP with Knowledge). A partial interpretation $\langle T, F \rangle$ over \mathcal{A} being total on X is a solution with assumptions $\langle T_Y, F_Y \rangle$ to the $\exists\forall$ ASP problem with knowledge $(P, \mathcal{A}, X, Y, Z)$ if $\langle T, F \rangle$ is a solution to the $\exists\forall$ ASP problem with knowledge $(P \cup T_Y \cup \{\leftarrow a \mid a \in F_Y\}, \mathcal{A}, X, Y - (T_Y \cup F_Y), Z \cup T_Y \cup F_Y)$.

Theorem 6. *Let $(P, \mathcal{A}, X, Y, Z)$ be an $\exists\forall$ ASP problem with knowledge, then for each stable model S of the logic program $E(\exists X\forall Y P) \cup K \cup \mathcal{A}$, $\langle \{a \mid t(a) \in S\}_{\mathcal{A}}, \{a \mid f(a) \in S\}_{\mathcal{A}} \rangle$ is a solution with assumptions $\langle \{a \mid t(a) \in S\}_Y, \{a \mid f(a) \in S\}_Y \rangle$ to the problem.*

6 Discussion

Starting from the basic problem of $\exists\forall$ ASP, we provided a series of characterizations of increasing complexity in order to capture concepts for planning with incomplete information. We addressed the various problems by means of a metaencoding that allows us to approximate the actual solutions. This is driven by the desire to stay within the realm of ASP’s computational complexity.

The current implementation deals with logic programs, whose decision problem is NP-complete and whose heads are free of quantified atoms. The latter are captured via predicates *exists/1* and *forall/1*. Predicate *kw/1* is used to represent know-whether atoms. The actual metaencoding generalizes the one described previously, and works for full *clingo* programs (and not just normalized ones). Alternatively, a metaencoding based in [22] can be used. The input program is reified by *clingo*, and passed along with the metaencodings once more to *clingo* for solving. The current implementation is available at <https://github.com/javier-romero/oclingo/tree/master/approximation> (but will move to <https://github.com/potassco>).

The approximation can be seen as an attempt to generalize [20] from action theories to ASP; it is closely related to the work on FO(ID) of [22]. The knowledge part generalizes the approach of [7] from action language SAS to ASP. The idea of planning with assumptions comes from [9]. They use an action language with conditional effects, and propose a translation to classical planning.

References

1. B. Andres, D. Rajaratnam, O. Sabuncu, and T. Schaub. Integrating ASP into ROS for reasoning in robots. In *LPNMR’15*, pages 69–82. Springer, 2015.
2. K. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, 1987.
3. M. Balduccini and M. Gelfond. Diagnostic reasoning with A-Prolog. *Theory and Practice of Logic Programming*, 3(4-5):425–461, 2003.
4. M. Balduccini and M. Gelfond. The autonomous agent architecture: An overview. In *Architectures for Intelligent Theory-Based Agents*, pages 1–6. AAAI Press, 2008.
5. C. Baral and M. Gelfond. Reasoning agents in dynamic domains. In *Logic-Based Artificial Intelligence*, pages 257–279. Kluwer, 2000.
6. C. Baral, V. Kreinovich, and R. Trejo. Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence*, 122:241–267, 2000.
7. M. Brenner and B. Nebel. Continual planning and acting in dynamic multiagent environments. *Autonomous Agents and Multi-Agent Systems*, 19(3):297–331, 2009.
8. W. Burgard, A. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, S. Thrun. The interactive museum tour-guide robot. In *AAAI’98*, pages 11–18. AAAI Press, 1998.

9. S. Davis-Mendelow, J. Baier, S. McIlraith. Assumption-based planning: Generating plans and explanations under incomplete knowledge. In *AAAI'13*, pages 209–216. AAAI Press, 2013.
10. T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. A logic programming approach to knowledge-state planning. *Artificial Intelligence*, 144(1-2):157–211, 2003.
11. M. Fitting. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 1985.
12. M. Gebser, R. Kaminski, and T. Schaub. Complex optimization in answer set programming. *Theory and Practice of Logic Programming*, 11(4-5):821–839, 2011.
13. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
14. H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–83, 1997.
15. V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39–54, 2002.
16. P. Simons, I. Niemelä, and T. Soinen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.
17. T. Son, C. Baral, T. Nam, and S. McIlraith. Domain-dependent knowledge in answer set planning. *ACM Transactions on Computational Logic*, 7(4):613–657, 2006.
18. T. Son, P. Tu, and C. Baral. Planning with sensing actions and incomplete information using logic programming. In *LPNMR'04*, pages 261–274. Springer, 2004.
19. P. Tu, T. Son, and C. Baral. 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, 2007.
20. P. Tu, T. Son, M. Gelfond, and A. Morales. Approximation of action theories and its application to conformant planning. *Artificial Intelligence*, 175(1):79–119, 2011.
21. A. Van Gelder, K. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
22. H. Vlaeminck, J. Vennekens, M. Denecker, and M. Bruynooghe. An approximative inference method for solving $\exists\forall$ SO satisfiability problems. *JAIR*, 45:79–124, 2012.