Planning with Incomplete Information in Quantified Answer Set Programming

Jorge Fandinno\textsuperscript{2,3} François Laferrière\textsuperscript{3} Javier Romero\textsuperscript{3} Torsten Schaub\textsuperscript{3} Tran Cao Son\textsuperscript{1}
\textsuperscript{1}New Mexico State University, USA \textsuperscript{2}Omaha State University, USA \textsuperscript{3}University of Potsdam, Germany

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

We present a general approach to planning with incomplete information in Answer Set Programming (ASP). More precisely, we consider the problems of conformant and conditional planning with sensing actions and assumptions. We represent planning problems using a simple formalism where logic programs describe the transition function between states, the initial states and the goal states. For solving planning problems, we use Quantified Answer Set Programming (QASP), an extension of ASP with existential and universal quantifiers over atoms that is analogous to Quantified Boolean Formulas (QBFs). We define the language of quantified logic programs and use it to represent the solutions to different variants of conformant and conditional planning. On the practical side, we present a translation-based QASP solver that converts quantified logic programs into QBFs and then executes a QBF solver, and we evaluate experimentally the approach on conformant and conditional planning benchmarks. Under consideration for acceptance in TPLP.

1 Introduction

We propose a general and uniform framework for planning in Answer Set Programming (ASP; Lifschitz 2002). Apart from classical planning, referring to planning with deterministic actions and complete initial states, our focus lies on conformant and conditional planning. While the former extends the classical setting by incomplete initial states, the latter adds sensing actions and conditional plans. Moreover, we allow for making assumptions to counterbalance missing information.

To illustrate this, let us consider the following example. There is a cleaning robot in a corridor going through adjacent rooms that may be occupied by people. The robot can go to the next room and can sweep its current room to clean it, but it should not sweep a room if it is occupied. We assume that nothing changes if the robot tries to go further than the last room, or if it sweeps a room that is already clean. The goal of the robot is to clean all rooms that are not occupied by people. We present a solution for any number of rooms, but in our examples we consider only two.

Classical planning. Consider an initial situation where the robot is in the first room, only the first room is clean, and no room is occupied. In this case, the classical planning problem is to find a plan that, applied to the initial situation, achieves the goal. The plan, where the robot goes to the second room and then sweeps it, solves this problem.

Conformant planning. Consider now that the robot initially does not know whether the rooms are clean or not. There are four possible initial situations, depending on the state of cleanliness of the two rooms. In this case, the conformant planning problem is
to find a plan that, applied to all possible initial situations, achieves the goal. The plan, where the robot sweeps, goes to the second room and sweeps it, solves that problem.

So far rooms were unoccupied. Now consider that initially the robot knows that exactly one of the rooms is occupied, but does not know which. Combining the previous four options about cleanliness with the two new options about occupancy, there are eight possible initial situations. It turns out that there is no conformant plan for this problem. The robot would have to leave the occupied room as it is and sweep the other, but there is no way of doing that without knowing which is the occupied room.

Assumption-based planning. At this point, the robot can try to make assumptions about the unknown initial situation, and find a plan that at least works under these assumptions, hoping that they will indeed hold in reality. In this case, a conformant planning problem with assumptions is to find a plan and a set of assumptions such that the assumptions hold in some possible initial situation, and the plan, applied to all possible initial situations satisfying the assumptions, achieves the goal. Assuming that room one is occupied, the plan where the robot goes to room two and then sweeps it, solves the problem. Another solution is to assume that the second room is occupied and simply sweep the first room.

Conditional planning. The robot has a safer approach, if it can observe the occupancy of a room and prepare a different subplan for each possible observation. This is similar to conformant planning, but now plans have actions to observe the world and different subplans for different observations. In our example, there is a conditional plan where the robot first observes if the first room is occupied, and if so, goes to the second room and sweeps it, otherwise it simply sweeps the first room. The robot could also make some assumptions about the initial situation, but this is not needed in our example.

Unfortunately, the expressiveness of regular ASP is insufficient to capture this variety of planning problems. While bounded classical and conformant planning are still expressible since their corresponding decision problems are still at the first and second level of the polynomial hierarchy, bounded conditional planning is $\text{PSPACE}$-complete (Turner 2002).

To match this level of complexity, we introduce a quantified extension of ASP, called Quantified Answer Set Programming (QASP), in analogy to Quantified Boolean Formulas (QBFs). More precisely, we start by capturing the various planning problems within a simple uniform framework centered on the concept of transition functions, mainly by retracing the work of Son et al. (2005; 2007; 2011). The core of this framework consists of a general yet simplified fragment of logic programs that aims at representing transition systems, similar to action languages (Gelfond and Lifschitz 1998) and (present-centered) temporal logic programs (Cabalar et al. 2018). We then extend the basic setting of ASP with quantifiers and define quantified logic programs, in analogy to QBFs. Although we apply QASP to planning problems, it is of general nature. This is just the same with its implementation, obtained via a translation of quantified logic programs to QBFs. This allows us to represent the above spectrum of planning problems by quantified logic programs and to compute their solutions with our QASP solver. Interestingly, the core planning problems are described by means of the aforementioned simple language fragment, while the actual type of planning problem is more or less expressed via quantification. Finally, we empirically evaluate our solver on conformant and conditional planning benchmarks.
2 Background

We consider normal logic programs over a set \( A \) of atoms with choice rules and integrity constraints. A rule \( r \) has the form \( H \leftarrow B \) where \( B \) is a set of literals, and \( H \) is either an atom \( p \), and we call \( r \) a normal rule, or \{\( p \)\} for some atom \( p \), making \( r \) a choice rule, or \( \bot \), so that \( r \) an integrity constraint. We usually drop braces from rule bodies \( B \), and also use \( l, B \) instead of \{\( l \)\} \( \cup B \) for a literal \( l \). We also abuse notation and identify sets of atoms \( X \) with sets of facts \{\( p \leftarrow \mid p \in X \)\}. A (normal) logic program is a set of (normal) rules. As usual, rules with variables are viewed as shorthands for the set of their ground instances.

We explain further practical extensions of ASP, like conditional literals and cardinality constraints, in passing. Semantically, we identify a body \( B \) with the conjunction of its literals, the head of a choice rule \{\( p \)\} with the disjunction \( p \lor \neg p \), a rule \( H \leftarrow B \) with the implication \( B \rightarrow H \), and a program with the conjunction of its rules. A set of atoms \( X \subseteq A \) is a stable model of a logic program \( P \) if it is a subset-minimal model of the formula that results from replacing in \( P \) any literal by \( \bot \) if it is not satisfied by \( X \). We let \( SM(P) \) stand for the set of stable models of \( P \).

The dependency graph of a logic program \( P \) has nodes \( A \), an edge \( p \rightarrow q \) if there is a rule whose head is either \( q \) or \{\( q \)\} and whose body contains \( p \), and an edge \( p \rightarrow q \) if there is a rule with head \( q \) or \{\( q \)\}, and \( \neg p \) in its body. A logic program is stratified if its dependency graph has no cycle involving a negative edge (\( \rightarrow \)). Note that stratified normal programs have exactly one stable model, unlike more general stratified programs.

ASP rests on a Generate-define-test (GDT) methodology \cite{Lifschitz2002}. Accordingly, we say that a logic program \( P \) is in GDT form if it is stratified and all choice rules in \( P \) are of form \{\( p \)\} \( \leftarrow \) such that \( p \) does not occur in the head of any other rule in \( P \). In fact, GDT programs constitute a normal form because every logic program can be translated into GDT form by using auxiliary atoms \cite{Niemelä2008,Fandinno2020}.

Quantified Boolean formulas (QBFs; \cite{Giunchiglia2009}) extend propositional formulas by existential (\( \exists \)) and universal (\( \forall \)) quantification over atoms. We consider QBFs over \( A \) of the form

\[
Q_0X_0 \ldots Q_nX_n \phi
\]

where \( n \geq 0 \), \( X_0, \ldots, X_n \) are pairwise disjoint subsets of \( A \), every \( Q_i \) is either \( \exists \) or \( \forall \), and \( \phi \) is a propositional formula over \( A \) in conjunctive normal form (CNF). QBFs as in (1) are in prenex conjunctive normal form. More general QBFs can be transformed to this form in a satisfiability-preserving way \cite{Giunchiglia2009}. Atoms in \( X_i \) are existentially (universally) quantified if \( Q_i \) is \( \exists \) (\( \forall \)). Sequences of quantifiers and sets \( Q_0X_0 \ldots Q_nX_n \) are called prefixes, and abbreviated by \( Q \). With it, a QBF as in (1) can be written as \( Q\phi \). As usual, we represent CNF formulas as sets of clauses, and clauses as sets of literals.

For sets \( X \) and \( Y \) of atoms such that \( X \subseteq Y \), we define \( fixbf(X,Y) \) as the set of clauses \{\{\( p \)\} \mid p \in X\} \( \cup \) \{\{\neg p\} \mid p \in Y \setminus X\} that selects models containing the atoms in \( X \) and no other atom from \( Y \). That is, if \( \phi \) is a formula then the models of \( \phi \cup fixbf(X,Y) \) are \{\( M \mid M \) is a model of \( \phi \) and \( M \cap Y = X \)\}. Given that a CNF formula is satisfiable if it has some model, the satisfiability of a QBF can be defined as follows:

- \( \exists X \phi \) is satisfiable if \( \phi \cup fixbf(Y,X) \) is satisfiable for some \( Y \subseteq X \).
- \( \forall X \phi \) is satisfiable if \( \phi \cup fixbf(Y,X) \) is satisfiable for all \( Y \subseteq X \).
- \( \exists X Q\phi \) is satisfiable if \( Q(\phi \cup fixbf(Y,X)) \) is satisfiable for some \( Y \subseteq X \).
\(\forall X \mathcal{Q}\phi\) is satisfiable if \(\mathcal{Q}(\phi \cup \mathit{fixfb}(Y, X))\) is satisfiable for all \(Y \subseteq X\).

The formula \(\phi\) in \(\mathcal{Q}\phi\) generates a set of models, while the prefix \(\mathcal{Q}\) can be interpreted as a kind of query over them. Consider \(\phi_1 = \{(a, b, \neg c), \{c\}\}\) and its models \(\{\{a, c\}, \{a, b, c\}, \{b, c\}\}\). Adding the prefix \(\mathcal{Q}_1 = \exists\{a\}\forall\{b\}\) amounts to querying if there is some subset \(Y_1\) of \(\{a\}\) such that for all subsets \(Y_2\) of \(\{b\}\) there is some model of \(\phi_1\) that contains the atoms in \(Y_1 \cup Y_2\) and no other atoms from \(\{a, b\}\). The answer is yes, for \(Y_1 = \{a\}\), hence \(\mathcal{Q}_1\phi_1\) is satisfiable. One can check that letting \(\mathcal{Q}_2\) be \(\exists\{b, c\}\forall\{a\}\) it holds that \(\mathcal{Q}_2\phi_1\) is satisfiable, while letting \(\mathcal{Q}_3\) be \(\exists\{a\}\forall\{b, c\}\) we have that \(\mathcal{Q}_3\phi_1\) is not.

### 3 Planning Problems

In this section, we define different planning problems with deterministic and non-concurrent actions using a transition function approach building on the work of Tu et al. (2007).

The domain of a planning problem is described in terms of fluents, i.e., properties changing over time, normal actions, and sensing actions for observing fluent values. We represent them by disjoint sets \(F\), \(A_n\), and \(A_s\) of atoms, respectively, let \(A\) be the set \(A_n \cup A_s\) of actions, and assume that \(F\) and \(A\) are non-empty. For clarity, we denote sensing actions in \(A_s\) by \(a^f\) for some \(f \in F\), indicating that \(a^f\) observes fluent \(f\). To simplify the presentation, we assume that sets \(F, A_n, A_s\) and \(A\) are fixed. A state \(s\) is a set of fluents, \(s \subseteq F\), that represents a snapshot of the domain. To describe planning domains, we have to specify what is the next state after the application of actions. Technically, this is done by a transition function \(\Phi\), that is, a function that takes as arguments a state and an action, and returns either one state or the bottom symbol \(\bot\). Formally, \(\Phi: \mathcal{P}(F) \times A \rightarrow \mathcal{P}(F) \cup \{\bot\}\), where \(\mathcal{P}(F)\) denotes the power set of \(F\). The case where \(\Phi(s, a) = \bot\) represents that action \(a\) is not executable in state \(s\).

**Example.** Let \(R\) be the set of rooms \(\{1, \ldots, r\}\). We represent our example domain with the fluents \(F = \{\mathit{at}(x), \mathit{clean}(x), \mathit{occupied}(x) \mid x \in R\}\), normal actions \(A_n = \{\mathit{go}, \mathit{sweep}\}\), and sensing actions \(A_s = \{\mathit{sense(occupied}(x)) \mid x \in R\}\). For \(r = 2, s_1 = \{\mathit{at}(1), \mathit{clean}(1)\}\) is the state representing the initial situation of our classical planning example. The transition function \(\Phi_e\) can be defined as follows: \(\Phi_e(s, \mathit{go})\) is \((s \setminus \{\mathit{at}(x) \mid x \in R\}) \cup \{\mathit{at}(x+1) \mid \mathit{at}(x) \in s, x < r\} \cup \{\mathit{at}(r) \mid \mathit{at}(r) \in s\}\), \(\Phi_e(s, \mathit{sweep})\) is \(s \cup \{\mathit{clean}(x) \mid \mathit{at}(x) \in s\}\) if, for all \(x \in R\), \(\mathit{at}(x) \in s\) implies \(\mathit{occupied}(x) \notin s\), and is \(\bot\) otherwise; and, for all \(x \in R\), \(\Phi_e(s, \mathit{sense(occupied}(x)))\) is \(s\) if \(\mathit{at}(x) \in s\) and is \(\bot\) otherwise.

Once we have fixed the domain, we can define a planning problem as a tuple \(\langle \Phi, I, G, A \rangle\) where \(\Phi\) is a transition function, and \(I, G \subseteq \mathcal{P}(F)\) are non-empty sets of initial and goal states. A conformant planning problem is a planning problem with no sensing actions, viz. \(A_s = \emptyset\), and a classical planning problem is a conformant planning problem where \(I\) is a singleton. A planning problem with assumptions is then a tuple \(\langle \Phi, I, G, A_s \rangle\) where \(\langle \Phi, I, G, A \rangle\) is a planning problem and \(A_s \subseteq F\) is a set of possible assumptions.

**Example.** The initial situation is \(I_1 = \{s_1\}\) for our classical planning problem, \(I_2 = \{\{\mathit{at}(1)\} \cup X \mid X \subseteq \{\mathit{clean}(1), \mathit{clean}(2)\}\}\) for the first conformant planning problem and \(I_3 = \{X \cup Y \mid X \in I_2, Y \in \{\mathit{occupied}(1)\}, \mathit{occupied}(2)\}\})\) for the second one. All problems share the goal states \(G_e = \{s \subseteq F \mid \text{for all } x \in R \text{ either } \mathit{occupied}(x) \in s \text{ or } \mathit{clean}(x) \in s\}\). Let \(\mathcal{P}_1\) be \(\langle \Phi_e, I_1, G_e\rangle\), \(\mathcal{P}_2\) be \(\langle \Phi_e, I_2, G_e\rangle\), and \(\mathcal{P}_3\) be \(\langle \Phi_e, I_3, G_e\rangle\). If we disregard sensing actions (and adapt \(\Phi_e\) consequently) these problems correspond to
our examples of classical and conformant planning, respectively. The one of assumption-based planning is given by $\mathcal{PP}_4 = (\Phi, I_3, G_c, \{\text{occupied}(1), \text{occupied}(2)\})$, and the one of conditional planning by $\mathcal{PP}_3$ with sensing actions.

Our next step is to define the solutions of a planning problem. For this, we specify what is a plan and extend transitions functions to apply to plans and sets of states. A plan starting from all initial states consistent with the assumptions. For example, the are consistent with some initial state, and condition (2) checks that $F_{\langle J \subseteq T, F \rangle}$ solves $PP_{\langle \Phi, I, G \rangle}$. For assumption-based planning, a tuple $(p, T, F)$, where $p$ is a plan and $T, F \subseteq A_S$ is a solution to a planning problem with assumptions $(\Phi, I, G, As)$ if (1) $J = \{s \mid s \in I, T \subseteq s, s \cap F = \emptyset\}$ is not empty, and (2) $p$ solves the planning problem $(\Phi, J, G)$. Condition (1) guarantees that the true assumptions $T$ and the false assumptions $F$ are consistent with some initial state, and condition (2) checks that $p$ achieves the goal starting from all initial states consistent with the assumptions. For example, the
planning problem with assumptions $\mathbb{PP}_4$ is solved by $\langle p_1, \{\text{occupied}(1)\}, \{\} \rangle$ and by $\langle \text{sweep}, \{\text{occupied}(2)\}, \{\} \rangle$.

### 4 Representing Planning Problems in ASP

In this section we present an approach for representing planning problems using logic programs. Let $F, A_n, A_s$ and $A$ be sets of atoms as before, and let $F' = \{ f' \mid f \in F \}$ be a set of atoms that we assume to be disjoint from the others. We use the atoms in $F'$ to represent the value of the fluents in the previous situation. We represent planning problems by planning descriptions, that consist of dynamic rules to represent transition functions, initial rules to represent initial states, and goal rules to represent goal states.

Formally, a dynamic rule is a rule whose head atoms belong to $F$ and whose body atoms belong to $A \cup F \cup F'$, an initial rule is a rule whose atoms belong to $F$, and a goal rule is an integrity constraint whose atoms belong to $F$. Then a planning description $\mathcal{D}$ is a tuple $(\mathcal{D}_R, \mathcal{I}_R, \mathcal{G}_R)$ of dynamic rules $\mathcal{D}_R$, initial rules $\mathcal{I}_R$, and goal rules $\mathcal{G}_R$. By $\mathcal{D}(\mathcal{D}), \mathcal{I}(\mathcal{D})$ and $\mathcal{G}(\mathcal{D})$ we refer to the elements $\mathcal{D}_R, \mathcal{I}_R$ and $\mathcal{G}_R$, respectively.

**Example.** We represent the transition function $\Phi_e$ by the following dynamic rules $\mathcal{D}_R$:

$\begin{align*}
  \text{at}(R) &\leftarrow \text{go}, \text{at}(R-1)', \text{R} < \text{r} & \text{clean}(R) &\leftarrow \text{sweep}, \text{at}(R)'\\
  \text{at}(r) &\leftarrow \text{go}, \text{at}(r)' & \text{clean}(R) &\leftarrow \text{clean}(R)'\\
  \text{at}(R) &\leftarrow \neg\text{go}, \text{at}(R)' & \text{occupied}(R) &\leftarrow \text{occupied}(R)'\\
  \bot &\leftarrow \text{sweep}, \text{at}(R)', \text{occupied}(R)' & \bot &\leftarrow \text{sense(occupied}(R)), \neg\text{at}(R)'
\end{align*}$

On the left column, the normal rules describe the position of the robot depending on its previous position and the action go, while the integrity constraint below forbids the robot to sweep if it is in a room that is occupied. On the right column, the first two rules state when a room is clean, the third one expresses that rooms remain occupied if they were before, and the last integrity constraint forbids the robot to observe a room if it is not at it. For the initial states, $I_1$ is represented by the initial rules $\mathcal{I}_R = \{ \text{at}(1) \leftarrow; \text{clean}(1) \leftarrow \}$, $I_2$ is represented by $\mathcal{I}_R$:

$\text{at}(1) \leftarrow \{ \text{clean}(R) \} \leftarrow \text{R} = 1..r$

and $I_3$ by $\mathcal{I}_R$, that contains the rules in $\mathcal{I}_R$ and also these ones:

$\{ \text{occupied}(R) \} \leftarrow \text{R} = 1..r \quad \bot \leftarrow \{ \text{occupied}(R) \} \neq 1$

The choice rules generate the different possible initial states, and the integrity constraint enforces that exactly one room is occupied. The goal states $G$ are represented by $\mathcal{G}_R = \{ \bot \leftarrow \neg\text{occupied}(R), \neg\text{clean}(R), \text{R} = 1..n \}$ that forbids states where some room is not occupied and not clean.

Next we specify formally the relation between planning descriptions and planning problems. We extract a transition function $\Phi(s, a)$ from a set of dynamic rules $\mathcal{D}_R$ by looking at the stable models of the program $s' \cup \{ a \leftarrow \} \cup DR$ for every state $s$ and action $a$, where $s'$ is $\{ f' \mid f \in s \}$. The state $s$ is represented as $s'$ to stand for the previous situation, and the action and the dynamic rules generate the next states. Given that we consider only deterministic transition functions, we restrict ourselves to deterministic problem descriptions, where we say that a set of dynamic rules $\mathcal{D}_R$ is deterministic if for every state $s \subseteq F$ and action $a \in A$, the program $s' \cup \{ a \leftarrow \} \cup DR$ has zero or one
where $n$ with assumptions

QLP is defined as follows: Given that a logic program is satisfiable if it has a stable model, the satisfiability of a program $P$ stable models of $X$ the atoms in $X$ set of rules $Q$ in (2) by $Q$. A quantified logic program over $A$ programs (QLPs), analogous to the extension of propositional formulas by QBFs. A Quantified Answer Set Programming (QASP) is an extension of ASP to quantified logic programs.

$PP$ represents $D$ the planning description presents the planning problem $⟨D⟩$ put together, we say that a deterministic and inertial planning description is inertial if for every state $s$ to planning problems in Section [8] easier. Formally, we say that a set of dynamic rules $DR$ is persistent. This condition is not strictly needed, but it makes the formulation of the solutions to deterministic and inertial planning descriptions.

Coming back to the initial and goal rules of a planning description $D$, the first ones represent the initial states $SM(I(D))$, while the second ones represent the goal states $SM(\{f ← f ∈ F\} ∪ G(D))$. In the latter case, the choice rules generate all possible states while the integrity constraints in $G(D)$ eliminate those that are not goal states. Given that we consider only non-empty subsets of initial and goal states, we require the programs $I(D)$ and $\{f ← f ∈ F\} ∪ G(D)$ to have at least one stable model. Finally, putting all together, we say that a deterministic and inertial planning description $D$ represents the planning problem $(Φ_D(D), SM(I(D)), SM(\{f ← f ∈ F\} ∪ G(D)))$. Moreover, the planning description $D$ together with a set of atoms $As$ represent the planning problem with assumptions $(Φ_D(D), SM(I(D)), SM(\{f ← f ∈ F\} ∪ G(D)), As)$.

Example. One can check that the dynamic rules $DR_e$ are deterministic, inertial, and define the transition function $Φ_e$ of the example. Moreover, $D_1 = ⟨DR_e, IR_1, GR_e⟩$ represents $PP_1$, $D_2 = ⟨DR_e, IR_2, GR_e⟩$ represents $PP_2$, $D_3 = ⟨DR_e, IR_3, GR_e⟩$ represents $PP_3$, and $D_3$ with the set of atoms $\{occupied(1), occupied(2)\}$ represents $PP_4$.

5 Quantified Answer Set Programming

Quantified Answer Set Programming (QASP) is an extension of ASP to quantified logic programs (QLPs), analogous to the extension of propositional formulas by QBFs. A quantified logic program over $A$ has the form

$$Q_0 X_0 \ldots Q_n X_n P \tag{2}$$

where $n ≥ 0$, $X_0, \ldots, X_n$ are pairwise disjoint subsets of $A$, every $Q_i$ is either $\exists$ or $\forall$, and $P$ is a logic program over $A$. Prefixes are the same as in QBFs, and we may refer to a QLP as in [2] by $QP$. For sets $X$ and $Y$ of atoms such that $X ⊆ Y$, we define $fixcons(X,Y)$ as the set of rules $\{⊥ ← ¬x \mid x ∈ X\} ∪ \{⊥ ← x \mid x ∈ Y \setminus X\}$ that selects stable models containing the atoms in $X$ and no other atom from $Y$. That is, if $P$ is a logic program then the stable models of $P \cup fixcons(X,Y)$ are $\{M \mid M$ is a stable model of $P$ and $M \cap Y = X\}$. Given that a logic program is satisfiable if it has a stable model, the satisfiability of a QLP is defined as follows:

- $∃XP$ is satisfiable if program $P \cup fixcons(Y,X)$ is satisfiable for some $Y ⊆ X$.
- $∀XP$ is satisfiable if program $P \cup fixcons(Y,X)$ is satisfiable for all $Y ⊆ X$.
• $\exists X QP$ is satisfiable if program $Q(P \cupfixcons(Y, X))$ is satisfiable for some $Y \subseteq X$.
• $\forall X QP$ is satisfiable if program $Q(P \cupfixcons(Y, X))$ is satisfiable for all $Y \subseteq X$.

As with QBFs, program $P$ in $QP$ generates a set of stable models, while its prefix $Q$ can be seen as a kind of query on it. Consider $P_1 = \{\{a\} \leftarrow \{b\} ; c \leftarrow a ; c \leftarrow b ; \bot \leftarrow \neg c\}$ and its stable models $\{a, c\}, \{a, b, c\}, \{b, c\}$. The prefixes of $Q_1P_1, Q_2P_1$ and $Q_3P_1$ pose the same queries over the stable models of $P$ than those posed in $Q_1\phi_1, Q_2\phi_1$ and $Q_3\phi_1$ over the models of $\phi_1$. Given that the stable models of $P_1$ and the models of $\phi_1$ coincide, the satisfiability of the $Q_iP_1$’s is the same as that of the corresponding $Q_i\phi_1$’s. This result is generalized by the following theorem that relates QLPs and QBFs.

**Theorem 5.1**

Let $P$ be a logic program over $A$ and $\phi$ be a CNF formula over $A \cup B$ such that $SM(P) = \{M \cap A \mid M$ is a model of $\phi\}$. For every prefix $Q$ whose sets belong to $A$, the QLP $QP$ is satisfiable if and only if the QBF $Q\phi$ is satisfiable.

The proof is by induction on the number of quantifiers in $Q$. The condition $SM(P) = \{M \cap A \mid M$ is a model of $\phi\} \subseteq A \cup B$ satisfies the condition of Theorem 5.1 using the tools $lp2normal, lp2acyc,$ and $lp2sat$ and then uses a QBF solver to decide the satisfiability of $Q\phi$. If $Q\phi$ is satisfiable and the outermost quantifier is existential, then the system returns an assignment to the atoms occurring in the scope of that quantifier.

### 6 Solving Planning Problems in QASP

In this section, we describe how to solve planning problems represented by a planning description using QASP.

**Classical planning.** We start with a planning description $\mathcal{P}P = \langle \Phi, \{s_0\}, G \rangle$. Our task, given a positive integer $n$, is to find a plan $[a_1; \ldots; a_n]$ of length $n$ such that $\Phi(\{s_0\}, p) \subseteq G$. This can be solved as usual in answer set planning (Lifschitz 2002), using choice rules to generate possible plans, initial rules to define the initial state of the problem, dynamic rules replicated $n$ times to define the next $n$ steps, and goal rules to check the goal conditions at the last step. To do that

1. [https://github.com/potassco/qasp2qbf](https://github.com/potassco/qasp2qbf)
2. [http://research.ics.aalto.fi/software/asp](http://research.ics.aalto.fi/software/asp)
in this context, we first let Domain be the union of the following sets of facts asserting the time steps of the problem, the actions, and the fluents that are sensed by each sensing action: \( \{ t(T) \mid t \in \{1, \ldots , n\} \} \), \( \{ \text{action}(a) \mid a \in A \} \), and \( \{ \text{senses}(a^f, f) \mid a^f \in A_s \} \). The last set is only needed for conditional planning, but we already add it here for simplicity. Given these facts, the following choice rule generates the possible plans of the problem:

\[
\{ \text{occ}(A, T) : \text{action}(A) \} \equiv 1 \leftarrow t(T)
\]

Additionally, by \( D^*_1 \) (\( D^*_G \), respectively) we denote the set of rules that results from replacing in \( I(D) \) (\( G(D) \), respectively) the atoms \( f \in F \) by \( h(f, 0) \) (by \( h(f, n) \), respectively); by \( D^*_B \) we denote the set of rules that results from replacing in \( D(D) \) the atoms \( f \in F \) by \( h(f, T) \), the atoms \( f' \in F' \) by \( h(f, T-1) \), the atoms \( a \in A \) by \( \text{occ}(a, T) \) and adding the atom \( t(T) \) to the body of every rule; and by \( D^\circ \) we denote the program \( D^*_1 \cup D^*_B \cup D^*_G \).

Putting all together, the program Domain \( \cup \) \( \{3\} \cup D^\circ \) represents the solutions to the planning problem \( \mathcal{PP} \). The choice rule \( \{3\} \) guesses plans \( [a_1; \ldots ; a_n] \) using atoms of the form \( \text{occ}(a_1, 1), \ldots , \text{occ}(a_n, n) \), the rules of \( D^*_1 \) define the initial state \( s_0 \) using atoms of the form \( h(\cdot , 0) \), the rules of \( D^*_B \) define the next states \( s_i = \Phi(s_{i-1}, a_i) \) for \( i \in \{1, \ldots , n\} \) using atoms of the form \( h(\cdot , i) \) while at the same time check that the actions \( a_i \) are executable in \( s_{i-1} \), and the rules of \( D^*_G \) check that the last state \( s_n \) belongs to \( G \). Of course, all this works only because \( D \) represents \( \mathcal{PP} \) and therefore \( \Phi \), \( \{s_0\} \) and \( G \) are defined by \( D(D) \), \( I(D) \) and \( G(D) \), respectively. Finally, letting \( \text{Occ} \) be the set of atoms \( \{ \text{occ}(a, t) \mid a \in A, t \in \{1, \ldots , n\} \} \), we can represent the solutions to \( \mathcal{PP} \) by the quantified logic program

\[
\exists \text{Occ}(\text{Domain} \cup \{3\} \cup D^\circ)
\]

where the atoms selected by the existential quantifier correspond to solutions to \( \mathcal{PP} \). Going back to our example, where \( D_1 \) represents the problem \( \mathcal{PP}_1 \), we have that for \( n = 2 \) the program \( \{1\} \) (adapted to \( D_1 \)) is satisfiable selecting the atoms \( \{ \text{occ}(go, 1), \text{occ}(sweep, 2) \} \) that represent the solution \( p_1 \).

**Conformant planning.** When \( D \) represents a conformant planning problem \( \mathcal{PP} = \langle \Phi, I, G \rangle \) our task is to find a plan \( p \) such that \( \Phi(I, p) \subseteq G \), or, alternatively, \( p \) must be such that for all \( s \in I \) it holds that \( \Phi(s, p) \subseteq G \). This formulation of the problem suggests to use a prefix \( \exists \forall \) where the existential quantifier guesses a plan \( p \) and the universal quantifier considers all initial states. Let us make this more concrete. From now on we assume that \( I(D) \) is in GDT form.\(^3\) If that is not the case then we can translate the program into GDT form using the method mentioned in the Background section, and we continue from there. Let Gen, Def and Test be the choice rules, normal rules and integrity constraints of \( D^*_1 \), respectively, and let Open be the set \( \{ h(f, 0) \mid \{ h(f, 0) \} \leftarrow \text{Gen} \} \) of possible guesses of the choice rules of \( D^*_1 \). Observe that for every possible set \( X \subseteq \text{Open} \) the program \( X \cup \text{Def} \) has a unique stable model \( M \), and if \( M \) is also a model of Test then \( M \) is a stable model of \( D^*_1 \). Moreover, note that all stable models of \( D^*_1 \) can be constructed in this manner. Given this, we say that the sets \( X \subseteq \text{Open} \) that lead to a stable model \( M \) of \( D^*_1 \) are relevant, because they can be used as representatives of the initial states, and the other sets in \( \text{Open} \) are irrelevant. Back to our quantified logic program, we are going to use the prefix \( \exists \text{Occ} \forall \text{Open} \). This works well with the logic program Domain \( \cup \{3\} \cup D^\circ \)

\(^3\) Note that this implies that \( D^*_2 \) is also in GDT form.
whenever all choices \( X \subseteq \text{Open} \) are relevant. But it fails as soon as there are irrelevant sets because, when we select them as subsets of \( \text{Open} \) in the universal quantifier, the resulting program becomes unsatisfiable. To fix this, we can modify our logic program so that for the irrelevant sets the resulting program becomes always satisfiable. We do that in two steps. First, we modify \( D^*_I \) so that the irrelevant sets lead to a unique stable model that contains the special atom \( \alpha(0) \). This is done by the program \( D^*_I \) that results from replacing in \( D^*_I \) the symbol \( \bot \) in the head of the integrity constraints by \( \alpha(0) \). Additionally, we consider the rule

\[
\alpha(T) \leftarrow t(T), \alpha(T-1)
\]

that derives \( \alpha(1), \ldots, \alpha(n) \) for the irrelevant sets. Second, we modify \( D^*_D \) and \( D^*_G \) so that whenever those special atoms are derived, these programs are immediately satisfied. This is done by the programs \( D^*_D \) and \( D^*_G \) that result from adding the literal \( \neg \alpha(T) \) to the bodies of the rules in \( D^*_D \) and \( D^*_G \), respectively. Whenever the special atoms \( \alpha(0), \ldots, \alpha(n) \) are derived, they deactivate the rules in \( D^*_D \) and \( D^*_G \) and make the program satisfiable. We denote by \( D^* \) the program \( D^*_I \cup D^*_D \cup D^*_G \). Then the following theorem establishes the correctness and completeness of the approach.

**Theorem 6.1**

Let \( D \) be a planning description that represents a conformant planning problem \( \mathcal{PP} \), and \( n \) be a positive integer. If \( I(P) \) is in \( \text{GDT} \) form, then there is a plan of length \( n \) that solves \( \mathcal{PP} \) if and only if the following quantified logic program is satisfiable:

\[
\exists \text{Occ} \forall \text{Open}(\text{Domain} \cup D^* \cup \mathbf{3} \cup (5))
\]

In our example, where \( D_2 \) represents the problem \( \mathcal{PP}_2 \), for \( n = 3 \) the program (adapted to \( D_2 \)) is satisfiable selecting the atoms \{\text{occ(sweep,1)}, \text{occ(go,2)}, \text{occ(sweep,3)}\} that represent the solution \( p_2 \), while for \( D_3 \), that represents \( \mathcal{PP}_3 \), the corresponding program is unsatisfiable for any integer \( n \).

**Assumption-based planning.** Let \( D \), together with a set of atoms \( \text{As} \subseteq F \), represent a conformant planning problem with assumptions \( \mathcal{PP} = (\Phi, I, G, \text{As}) \). To solve this problem we have to find a plan \( p = [a_1; \ldots; a_n] \) and a set of assumptions \( T, F \subseteq \text{As} \) such that (1) the set \( J = \{s \mid s \in I, T \subseteq s, s \cap F = \emptyset\} \) is not empty, and (2) \( p \) solves the conformant planning problem \( (\Phi, J, G) \). The formulation of the problem suggests that first we can guess the set of assumptions \( T \) and \( F \), and then check (1) and (2) separately. The guess can be represented by the set of rules \( \text{Guess} \) that consists of the facts \{\text{assumable}(f) \mid f \in \text{As}\} and the choice rule

\[
\{\text{assume}(F, true); \text{assume}(F, false)\} \leq 1 \leftarrow \text{assumable}(F)
\]

that generates all possible sets of assumptions using the predicate \text{assume}/2. Moreover, we add the set of atoms \( \text{Assume} = \{\text{assume}(f, v) \mid f \in \text{As}, v \in \{true, false\}\} \) to \text{Occ} at the outermost existential quantifier of our program. Condition (1) can be checked by the set of rules \( C1 \) that can be divided in two parts. The first part is a copy of the initial rules, that consists of the rules that result from replacing in \( I(P) \) every atom \( f \in F \) by \( \text{init}(f) \), and in this way generates all possible initial states in \( I \) using the predicate \( \text{init}/1 \). The second part contains the integrity constraints

\[
\bot \leftarrow \neg \text{init}(F), \text{assume}(F, true) \quad \bot \leftarrow \text{init}(F), \text{assume}(F, false)
\]
that guarantee that the guessed assumptions represented by assume/2 hold in some initial state represented by init/1. Condition (2) can be represented extending the program for conformant planning with the following additional rules $C2$, stating that the initial states that do not agree with the guessed assumptions are irrelevant:

$$\alpha(0) \leftarrow \neg h(F,0), \text{assume}(F,\text{true}) \quad \alpha(0) \leftarrow h(F,0), \text{assume}(F,\text{false})$$

With these rules, the plans only have to succeed starting from the initial states that agree with the assumptions, and condition (2) is satisfied.

**Theorem 6.2**

Let $D$, $\mathcal{PP}$, and $A$s be as specified before, and $n$ be a positive integer. If $\mathbf{I}(D)$ is in $GDT$ form, then there is a plan with assumptions of length $n$ that solves $\mathcal{PP}$ if and only if the following quantified logic program is satisfiable:

$$\exists (\text{Occ} \cup \text{Assume}) \forall \text{Open} (\text{Domain} \cup D^* \cup (5) \cup (5) \cup \text{Guess} \cup C1 \cup C2)$$

(7)

In our example, where $D_3$ together with the set of atoms \{occupied(1), occupied(2)\} represents $\mathcal{PP}_4$, we have that for $n = 2$ the program (7) (adapted to $D_3$) is satisfiable selecting the atoms \{occ(go, 1), occ(sweep, 2), assume(occupied(1), true)\} that represent the solution $\langle p_1, \{\text{occupied}(1)\}, \{} \rangle$, and for $n = 1$ selecting the atoms \{occ(sweep, 1), assume(occupied(2), true)\} that represent the solution $\langle \text{sweep}, \{\text{occupied}(2)\}, \{} \rangle$.

**Conditional planning.** Consider the case where $D$ represents a planning problem with sensing actions $PP = \langle \Phi, I, G \rangle$. Again, we have to find a plan $p$ such that $\Phi(I,p) \subseteq G$, but this time $p$ has the form of a tree where sensing actions can be followed by different actions depending on sensing results. This suggests a formulation where we represent the sensing result at time point $T$ by the truth value of an atom obs(true, $T$), we guess those possible observations with a choice rule:

$$\{\text{obs(true, } T)\} \leftarrow t(T), T < n$$

(8)

and, letting $\text{Obs}_t$ be \{obs(true, $t$)\} and $\text{Occ}_t$ be \{occ($a$, $t$) | $a \in A$\} for $t \in \{1, \ldots, n\}$, we consider a QLP of the form

$$\forall \text{Occ}_1 \forall \text{Obs}_1 \ldots \forall \text{Occ}_{n-1} \forall \text{Obs}_{n-1} \exists \text{Occ}_n \forall \text{Open} (\text{Domain} \cup D^* \cup (5) \cup (5) \cup (8))$$

This formulation nicely allows for a different plan $[a_1; \ldots ; a_n]$ for every sequence $[O_1; \ldots ; O_{n-1}]$ of observations $O_i \subseteq \text{Obs}_t$. But at the same time it requires that each such plan achieves the goal for all possible initial situations, and this requirement is too strong. Actually, we only want the plans to work for those cases where, at every time step $T$, the result of a sensing action $a^f$, represented by obs(true, $T$), coincides with the value of the sensed fluent represented by $h(f, T)$. We can achieve this by signaling the other cases as irrelevant with the following rule:

$$\alpha(T) \leftarrow t(T), \text{occ}(A, T-1), \text{senses}(A, F), \{h(F, T-1), \text{obs(true, } T-1)\} = 1$$

(9)

where the cardinality constraint $\{h(F, T-1), \text{obs(true, } T-1)\} = 1$ holds if the truth value of obs(true, $T-1$) and $h(F, T-1)$ is not the same. Another issue with the previous QLP is that it allows normal actions at every time point $T$ to be followed by different actions at $T+1$ for each value of obs(true, $T$). This is not a problem for the correctness of the approach, but it is not a natural representation. To fix this, we can consider that, whenever
a normal action occurs at time point \( T \), the case where \( \text{obs}(true,T) \) holds is irrelevant:

\[
\alpha(T) \leftarrow t(T), \text{occ}(A,T), \{\text{senses}(A,F)\} = 0, \text{obs}(true,T)
\]  

(10)

Appart from this, note that in conditional planning the different subplans may have different lengths. For this reason, in the choice rule we have to replace the symbol ‘=’ by ‘\( \leq \)’. We denote the new rule by (3) ≤.

**Theorem 6.3**

Let \( D \) and \( \mathcal{PP} \) be as specified before, and \( n \) be a positive integer. If \( I(D) \) is in \( \text{GDT} \) form, then there is a plan of length less or equal than \( n \) that solves \( \mathcal{PP} \) if and only if the following quantified logic program is satisfiable:

\[
\exists \text{Occ}_1 \forall \text{Obs}_1 \ldots \exists \text{Occ}_{n-1} \forall \text{Obs}_{n-1} \exists \text{Occ}_n \forall \text{Open}(\text{Domain} \cup \mathcal{P} \cup \{\text{true}, \text{truth}, \text{true}\}) ... (11)
\]

(11)

For \( D_3 \), that represents the problem \( \mathcal{PP}_3 \), and \( n = 3 \), the program (11) (adapted to \( D_3 \)) is satisfiable selecting first \( \{\text{occ}\text{(sense(occupied(1)))}\} \) at \( \text{Occ}_1 \), then at \( \text{Occ}_2 \) selecting \( \{\text{occ}\text{(clean,2)}\} \) for the subset \( \{\} \subseteq \text{Obs}_3 \) and \( \{\text{occ}\text{(go,2)}\} \) for the subset \( \text{Obs}_1 \subseteq \text{Obs}_1 \), and finally at \( \text{Occ}_3 \) selecting \( \{\} \) in all cases except for the subsets \( \text{Obs}_1 \subseteq \text{Obs}_1 \) and \( \{\} \subseteq \text{Obs}_2 \) that we select \( \{\text{occ}\text{(clean,3)}\} \). This assignment corresponds to plan \( p_3 \).

### 7 Experiments

We evaluate the performance of QASP2QBF in conformant and conditional planning benchmarks. We consider the problem OPT of computing a plan of optimal length. To solve it we first run the solver for length 1, and successively increment the length by 1 until an optimal plan is found. The solving times of this procedure are usually dominated by the unsatisfiable runs. To complement this, we also consider the problem SAT of computing a plan of a fixed known length, for which we know that a solution exists.

We evaluate QASP2QBF 1.0 and combine it with 4 different QBF solvers: CAQE 4.0.14 (Rabe and Tenknap 2015), DEPQBF 6.03 (Lonsing and Egly 2017), QESTO 1.0 (Janota and Marques-Silva 2015) and QUTE 1.1 (Peitl et al. 2019); and either none or one of 3 preprocessors: BLOQWER v37 (Biere et al. 2011), HQSPRE 1.4 (H: Wimmer et al. 2017), and QATPRED 2.0 (Q: Lonsing and Egly 2019); for a total of 16 configurations. By CAQE we refer to the combination of QASP2QBF with CAQE without preprocessor, and by CAQEP to QASP2QBF with CAQE and the preprocessor BLOQWER. We proceed similarly for other QBF solvers and preprocessors. We compare QASP2QBF with the incomplete planner CPASP (Tu et al. 2007), that translates a planning description into a normal logic program that is fed to an ASP solver. In the experiments we have used the ASP solver CLINGO (version 5.5.4) and evaluated its 6 basic configurations (CRAFTY (C), FRUMPY (F), HANDY (H), JUMPY (J), TRENDY (R), and TWEETY (T)). We refer to CPASP with CLINGO and configuration CRAFTY as CLINGO, and similarly with the others.

We use the benchmark set from Tu et al. (2007), but we have increased the size of the instances of some domains if they were too easy. The conformant domains are six variants of Bomb in the Toilet (Bt, Bmt, Btc, Bntc, Bntuc, Bntuc), Domino and Ring. We have

---

4 We follow the selection of QBF solvers and preprocessors of Mayer-Eichberger and Saffidine (2020).

5 https://potassco.org/clingo
also added a small variation of the Ring domain, called Ringu, where the room of the agent is unknown, and the planner CPasp cannot find any plan due to its incompleteness.

The conditional domains are four variations of Bomb in the Toilet with Sensing Actions (Bts1, Bts2, Bts3 and Bts4), Domino, Medical Problem (Med), Ring and Sick. All domains have 5 instances of increasing difficulty, except Domino in conformant planning that has 6, and Ring in conditional planning that has 4. For the problem SAT, the fixed plan length is always the minimal plan length for CPasp.

All experiments ran on an Intel Xeon 2.20GHz processor under Linux. Each run was limited to 30 minutes runtime and 16 GB of memory. We report the aggregated results per domain: average runtime in seconds and number of timeouts in parentheses for OPT, next to the average runtime in seconds for SAT, for which there were very few timeouts.

To calculate the averages, we add 1800 seconds for every timeout. In Appendix A, we report these results for every solver and configuration, and provide further details. Here, we show and discuss the best configuration for each solver, separately for conformant planning in Table 1, and for conditional planning in Table 2.

In conformant planning, looking at the qasp2qbf configurations, for the variations of Bt, CAQE and QESTO perform better than DEPQBF and QUTE. Domino is solved very quickly by all solvers, while in Ring and Ringu CAQE clearly outperforms the others. The planner CLINGO, in the variations of Bt and Domino, in OPT has a similar performance to the best qasp2qbf solvers, while in SAT it is much faster and solves the problems in less than a second. In Ring, however, its performance is worse than that of CAQE. Finally, in Ringu for OPT, given the incompleteness of the system, it never manages to find a plan and always times out. In conditional planning, CAQE is the best for the variations of Bts, while CLINGO is better than the other solvers for SAT but worse for OPT. In Domino, for OPT, the QASP2QBF solvers perform better than CLINGO, while for SAT only QESTO matches its performance. Finally, for Med, Ring, and Sick, all solvers yield similar times.

Summing up, we can conclude that QASP2QBF with the right QBF solver and preprocessor compares well to CPasp, except for the SAT problem in conformant planning, while on the other hand it can solve problems, like Ringu, that are out of reach for CPasp due to its incompleteness.

|         | CLINGO | CAQE | DEPQBF | QESTO | QUTE |
|---------|--------|------|--------|-------|------|
| Bt      | 43 (0) | 0    | 46 (0) | 0     | 0    |
| Bnt     | 50 (0) | 0    | 155 (0)| 2     | 0    |
| Btc     | 258 (0)| 0    | 413 (1)| 1     | 566 (1)| 2    |
| Bntc    | 744 (2)| 0    | 771 (2)| 9     | 1083 (3)| 9    |
| Btuc    | 245 (0)| 0    | 400 (1)| 1     | 538 (1)| 1    |
| Bntuc   | 739 (2)| 0    | 780 (2)| 14    | 1085 (3)| 12   |
| Domino  | 0 (0)  | 0    | 0 (0)  | 0     | 0 (0) |
| Ring    | 673 (1)| 416  | 281 (0)| 40    | 1120 (3)| 1109 |
| Ringu   | 1800 (5)| 0   | 246 (0)| 50    | 1800 (5)| 1800 |
| Total   | 505 (10)| 46  | 343 (6)| 12    | 764 (17)| 326  |

Table 1: Experimental results on conformant planning
Conformant and conditional planning have already been addressed with ASP (Eiter et al. 2003; Son et al. 2005; Tu et al. 2007; Tu et al. 2011). Eiter et al. (2003) introduce the system \texttt{dlv}\textsc{K} for planning with incomplete information. It solves a conformant planning problem by first generating a potential plan and then verifying it, no sensing actions are considered. Son et al. (2005; 2007; 2011) propose an approximation semantics for reasoning about action and change in the presence of incomplete information and sensing actions. This is then used for developing ASP-based conformant and conditional planners, like \texttt{CPasp}, that are generally incomplete.

Closely related is SAT-based conformant planning: C-Plan (Castellini et al. 2003) is similar to \texttt{dlv}\textsc{K} in identifying a potential plan before validating it. \textsc{Compile-project-sat} (Palacios and Geffner 2005) uses a single call to a SAT-solver to compute a conformant plan. Their validity check is doable in linear time, if the planning problem is encoded in deterministic decomposable negation normal form. Unlike this, \textsc{QBFPlan} (Rintanen 1999) maps the problem into QBF and uses a QBF-solver as back-end.

A more recent use of ASP for computing conditional plans is proposed by Yalciner et al. (2017). The planner deals with sensing actions and incomplete information; it generates multiple sequential plans before combining them in a graph representing a conditional plan. Cardinality constraints, defaults, and choices are used to represent the execution of sensing actions, their effects, and branches in the final conditional plan. In addition, the system computes sequential plans in parallel and also avoids regenerating plans.

Assumption-based planning, as considered here, is due to Davis-Mendelow et al. (2013). In that work, the problem is solved by translating it into classical planning using an adaptation of the translation of Palacios and Geffner (2009). To the best of our knowledge, there exists no ASP-based planner for this type of problems.

There are a number of extensions of ASP to represent problems whose complexity lays beyond NP in the polynomial hierarchy. A comprehensive review was made by Amendola et al. (2019), that presents the approach that is closer to \textsc{Qasp}, named ASP with Quantifiers (ASP(Q)). Like \textsc{Qasp}, it introduces existential and universal quantifiers, but they range over stable models of logic programs and not over atoms. This quantification over stable models is very useful for knowledge representation. For example, it allows us to represent conformant planning problems without the need of additional $\alpha$ atoms, using the following ASP(Q) program:

$$\exists^e \big(D\texttt{omain} \cup \{3\}\big) \land \forall^e D_1^0 \exists^e \big(D_0^e \cup D_G^e\big)$$  \hspace{1cm} (12)
where $\exists^s$ and $\forall^s$ are existential and universal stable model quantifiers, respectively. The program is coherent (or satisfiable, in our terms) if there is some stable model $M_1$ of $\text{Domain} \cup \{3\}$ such that for all stable models $M_2$ of $M_1 \cup D^o_I$ there is some stable model of $M_2 \cup D^o_D \cup D^o_G$. Stable models $M_1$ correspond to possible plans. They are extended in $M_2$ by atoms representing initial states, that are used in $M_2 \cup D^o_D \cup D^o_G$ to check if the plans achieve the goal starting from all those initial states. Assumption-based planning can be represented in a similar way, while for conditional planning we have not been able to come up with any formulation that does not use additional $\alpha$ atoms.

As part of this work, we have developed translations between QASP and ASP($Q$). We leave their formal specification to Appendix B and illustrate them here for conformant planning. From ASP($Q$) to QASP, we assume that $D^o_I$ is in GDT form, and otherwise we translate it into this form. Then, the translation of an ASP($Q$) program of the form (12) yields a QLP that is essentially the same as (6), except for some renaming of the additional $\alpha$ atoms and some irrelevant changes in the prefix. In the other direction, the QLP program (6) is translated to the ASP($Q$) program

$$\exists^s P_0 \forall^s P_1 \exists^s (\text{Domain} \cup \{3\} \cup \{5\} \cup O)$$

where $P_0$ is $\{\{p'\} \leftarrow p \in \text{Occ}\}$, $P_1$ is $\{\{p'\} \leftarrow p \in \text{Open}\}$, and $O$ contains the set of rules $\bot \leftarrow p, \neg p'$ and $\bot \leftarrow \neg p, p'$ for every $p \in \text{Occ} \cup \text{Open}$. Programs $P_0$ and $P_1$ guess the values of the atoms of the prefix using additional atoms $p'$, and the constraints in $O$ match those guesses to the corresponding original atoms.

9 Conclusion

We defined a general ASP language to represent a wide range of planning problems: classical, conformant, with assumptions, and conditional with sensing actions. We then defined a quantified extension of ASP, viz. QASP, to represent the solutions to those planning problems. Finally, we implemented and evaluated a QASP solver, available at potassco.org, to compute the solutions to those planning problems. Our focus lays on the generality of the language and the tackled problems; on the formal foundations of the approach, by relating it to simple transition functions; and on having a baseline implementation, whose performance we expect to improve further in the future.

References

Amendola, G., Ricca, F., and Truszczynski, M. 2019. Beyond NP: quantifying over answer sets. Theory and Practice of Logic Programming 19, 5-6, 705–721.

Biere, A., Lonsing, F., and Seidl, M. 2011. Blocked clause elimination for QBF. In Proceedings of CADE’11, Lecture Notes in Computer Science, vol. 6803. Springer-Verlag, 101–115.

Cabalar, P., Kaminski, R., Schaub, T., and Schuhmann, A. 2018. Temporal answer set programming on finite traces. Theory and Practice of Logic Programming 18, 3-4, 406–420.

Castellini, C., Giunchiglia, E., and Tacchella, A. 2003. SAT-based planning in complex domains: Concurrency, constraints and nondeterminism. Artificial Intelligence 147, 1-2, 85–117.

Davis-Mendelow, S., Baier, J., and McIlraith, S. 2013. Assumption-based planning: Generating plans and explanations under incomplete knowledge. In Proceedings of AAAI’13, M. desJardins and M. Littman, Eds. AAAI Press, 209–216.
EITER, T., FABER, W., LEONE, N., PFEIFER, G., AND POLLERES, A. 2003. A logic programming approach to knowledge-state planning. *Artificial Intelligence* 144, 1-2, 157–211.

FANDINNO, J., MISHIRA, S., ROMERO, J., AND SCHAUB, T. 2020. Answer set programming made easy. In *Proceedings of ASPOCP’20*, M. Hecher and J. Zangari, Eds.

GELFOND, M. AND LIFSCHITZ, V. 1998. Action languages. *Electronic Transactions on Artificial Intelligence* 3, 6, 193–210.

GIUNCHIGLIA, E., MARIN, P., AND NARIZZANO, M. 2009. Reasoning with quantified Boolean formulas. In *Handbook of Satisfiability*, A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds. Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, Chapter 24, 761–780.

JANHUNEN, T. 2004. Representing normal programs with clauses. In *Proceedings of ECAI’04*, R. López de Mántaras and L. Saitta, Eds. IOS Press, 358–362.

JANOTA, M. AND MARQUES-SILVA, J. 2015. Solving QBF by clause selection. In *Proceedings of IJCAI’15*, Q. Yang and M. Wooldridge, Eds. AAAI Press, 325–331.

LIFSCHITZ, V. 2002. Answer set programming and plan generation. *Artificial Intelligence* 138, 1-2, 39–54.

LIFSCHITZ, V. AND TURNER, H. 1994. Splitting a logic program. In *Proceedings of the Eleventh International Conference on Logic Programming*. MIT Press, 23–37.

LONSING, F. AND EGLY, U. 2017. Depqbf 6.0: A search-based QBF solver beyond traditional QCDCL. In *Proceedings of CADE’17*. Springer-Verlag, 371–384.

LONSING, F. AND EGLY, U. 2019. Qratpre+: Effective QBF preprocessing via strong redundancy properties. In *Proceedings of SAT’19*. Springer-Verlag, 203–210.

MAYER-EICHBERGER, V. AND SAFFIDINE, A. 2020. Positional games and QBF: the corrective encoding. In *Proceedings of SAT’20*. Springer-Verlag, 447–463.

NIEMELÄ, I. 2008. Answer set programming without unstratified negation. In *Proceedings of ICLP’08*. Springer-Verlag, 88–92.

PALACIOS, H. AND GEFNER, H. 2005. Mapping conformant planning into SAT through compilation and projection. In *Proceedings of CAEPIA’05*. Springer-Verlag, 311–320.

PALACIOS, H. AND GEFNER, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research* 35, 623–675.

PETTL, T., SLIVOVSKY, F., AND SZIEDE, S. 2019. Dependency learning for QBF. *Journal of Artificial Intelligence Research* 65, 180–208.

RABE, M. N. AND TENTRUP, L. 2015. CAQE: A certifying QBF solver. In *Proceedings of FMCA/CAD’15*, R. Kaivola and T. Wahl, Eds. IEEE Computer Society Press, 136–143.

RINTANEN, J. 1999. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research* 10, 323–352.

SON, T., TU, P., GELFOND, M., AND MORALES, A. 2005. An approximation of action theories of AL and its application to conformant planning. In *Proceedings of LPNMR’05*. Springer-Verlag, 172–184.

TU, P., SON, T., AND 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, 4, 377–450.

TU, P., SON, T., GELFOND, M., AND MORALES, A. 2011. Approximation of action theories and its application to conformant planning. *Artificial Intelligence* 175, 1, 79–119.

TURNER, H. 2002. Polynomial-length planning spans the polynomial hierarchy. In *Proceedings of JELIA’02*. Springer-Verlag, 111–124.

WIMMER, R., REIMER, S., MARIN, P., AND BECKER, B. 2017. Hqspre – an effective preprocessor for QBF and DQBF. In *Proceedings of TACAS’17*. Springer-Verlag, 373–390.

YALCINER, I., NOUMAN, A., PATOGLU, V., AND ERDEM, E. 2017. Hybrid conditional planning using answer set programming. *Theory and Practice of Logic Programming* 17, 5-6, 1027–1047.
Appendix A Results

In this section, we provide some additional information about the experiments, and present the results for every solver and configuration.

We note that in conditional planning, CPasp receives as input an additional width parameter that determines the minimum number of branches of a conditional plan. QASP2QBF receives no information of that kind, and in this regard it solves a more difficult task than CPasp. On the other hand, that width parameter allows CPasp to return the complete conditional plan, while QASP2QBF only returns the assignment to the first action of the plan. Observe also that given that CPasp is incomplete, in some instances QASP2QBF can find plans for OPT whose length is smaller than the fixed length of SAT. This explains why in some few cases QASP2QBF solves faster the OPT problem than SAT.

In the experiments, the times for grounding and for the translators lp2normal and lp2sat are negligible, while, in the SAT problem, preprocessing takes on average 5 and 2 seconds in conformant and conditional planning, respectively. We expect these times to be similar for the OPT problem. The reported times are dominated by CLINGO’s solving time in CPasp, and by the QBF solvers in QASP2QBF. But the usage of preprocessors is fundamental for the performance of QASP2QBF. For example, in conformant planning, CAQE alone results in 18 timeouts, while with QRATPRE+ they go down to only 6.

|     | CLINGO^c | CLINGO^d | CLINGO^u | CLINGO^v | CLINGO^w | CLINGO^x |
|-----|-----------|-----------|-----------|-----------|-----------|-----------|
| Bt  | 52 (0)    | 176 (0)   | 0         | 362 (1)   | 0         | 363 (1)   |
| Bmt | 49 (0)    | 163 (0)   | 0         | 50 (0)    | 363 (1)   | 35 (0)    |
| Btc | 236 (0)   | 395 (1)   | 0         | 258 (0)   | 380 (1)   | 282 (0)   |
| Bmtc| 746 (2)   | 998 (2)   | 0         | 744 (2)   | 773 (2)   | 739 (2)   |
| Btuc| 367 (1)   | 389 (1)   | 0         | 245 (0)   | 382 (1)   | 380 (1)   |
| Bmtuc| 749 (2)  | 938 (2)   | 0         | 739 (2)   | 778 (2)   | 742 (2)   |
| Domino| 0 (0)    | 0 (0)     | 0         | 0 (0)     | 0 (0)     | 0 (0)     |
| Ring| 774 (2)   | 506 (1)   | 1102 (3)  | 777 (1)   | 673 (1)   | 645 (1)   |
| Ringu| 1800 (5)  | 1800 (5)  | 1800 (5)  | 1800 (5)  | 1800 (5)  | 1800 (5)  |

**Table A 1: Conformant planning with CLINGO**

|     | CAQE^c   | CAQE^d   | CAQE^u   | CAQE^v   |
|-----|-----------|-----------|-----------|-----------|
| Bt  | 69 (0)    | 3         | 377 (0)   | 1         |
| Bmt | 44 (0)    | 3         | 149 (0)   | 4         |
| Btc | 440 (1)   | 3         | 487 (1)   | 3         |
| Bmtc| 814 (2)   | 7         | 777 (2)   | 7         |
| Btuc| 476 (1)   | 3         | 475 (1)   | 3         |
| Bmtuc| 849 (2)  | 14        | 789 (2)   | 15        |
| Domino| 1500 (5) | 1500      | 0 (0)     | 0 (0)     |
| Ring| 1104 (3)  | 1097      | 219 (0)   | 30        |
| Ringu| 1630 (4)  | 1643      | 1018 (2)  | 969       |

**Table A 2: Conformant planning with CAQE**


## Table A 3: Conformant planning with depqbf

|      | depqbf | depqbf | depqbf | depqbf |
|------|--------|--------|--------|--------|
| Bt   | 740 (2) | 0      | 744 (2) | 1      |
| Bmt  | 974 (2) | 2      | 1089 (3) | 3    |
| Btc  | 1111 (3) | 3     | 1084 (3) | 3    |
| Bmtc | 1494 (4) | 5     | 1424 (4) | 7    |
| Bmtuc| 1102 (3) | 7     | 1088 (3) | 3    |
| Domino| 0 (0)   | 0      | 1 (0)   | 0     |
| Ring | 1800 (5) | 1800  | 1708 (4) | 1468 |
| Ringu| 1800 (5) | 1800  | 1800 (5) | 1800 |
| Total| 1166 (28) | 411   | 1155 (28) | 365  |

## Table A 4: Conformant planning with qesto

|      | qesto | qesto | qesto | qesto |
|------|-------|-------|-------|-------|
| Bt   | 725 (2) | 3     | 458 (1) | 1    |
| Bmt  | 1081 (3) | 9     | 443 (1) | 4    |
| Btc  | 1081 (3) | 7     | 777 (2) | 6    |
| Bmtc | 1440 (4) | 20    | 1104 (3) | 23   |
| Bmtuc| 1479 (4) | 82    | 1443 (4) | 8    |
| Domino| 0 (0)   | 0     | 1 (0)   | 0     |
| Ring | 1800 (5) | 1800  | 1708 (4) | 1468 |
| Ringu| 1800 (5) | 1800  | 1800 (5) | 1800 |
| Total| 1230 (31) | 516   | 728 (16) | 170  |

## Table A 5: Conformant planning with qute

|      | qute | qute | qute | qute |
|------|------|------|------|------|
| Bt   | 864 (2) | 3     | 1081 (3) | 1    |
| Bmt  | 1095 (3) | 9     | 1180 (3) | 3    |
| Btc  | 1095 (3) | 16    | 1093 (3) | 3    |
| Bmtc | 1453 (4) | 27    | 1444 (4) | 8    |
| Bmtuc| 1511 (4) | 55    | 1101 (3) | 3    |
| Domino| 0 (0)   | 0     | 0 (0)   | 0     |
| Ring | 1800 (5) | 1449  | 1702 (4) | 27   |
| Ringu| 1800 (5) | 1800  | 1800 (5) | 1800 |
| Total| 1328 (33) | 572   | 1238 (29) | 342  |

## Table A 6: Conditional planning with clingo

|      | clingo | clingo | clingo | clingo | clingo | clingo |
|------|--------|--------|--------|--------|--------|--------|
| Bts1 | 1041 (2) | 121   | 1080 (3) | 77   |
| Bts2 | 1080 (3) | 60    | 1080 (3) | 361  |
| Bts3 | 993 (2) | 85    | 1071 (2) | 215  |
| Bts4 | 1060 (2) | 69    | 1080 (3) | 115  |
| Domino| 872 (2) | 15    | 885 (2) | 59   |
| Med  | 0 (0)   | 0     | 0 (0)   | 0     |
| Ring | 902 (2) | 900   | 905 (2) | 900  |
| Sick | 0 (0)   | 0     | 0 (0)   | 0     |
| Total| 743 (13) | 156   | 762 (15) | 215  |

## Table A 7: Conditional planning with clingo
### Table A 7: Conditional planning with CAQE

|       | CAQE^a | CAQE^b | CAQE^c | CAQE^d |
|-------|---------|---------|---------|---------|
| Bts1  | 370 (1) | 368     | 14 (0)  | 3       | 16 (0)  | 3       | 366 (1) | 363     |
| Bts2  | 378 (1) | 376     | 14 (0)  | 3       | 15 (0)  | 4       | 106 (0) | 54      |
| Bts3  | 372 (1) | 379     | 15 (0)  | 4       | 19 (0)  | 7       | 135 (0) | 65      |
| Bts4  | 374 (1) | 384     | 17 (0)  | 4       | 25 (0)  | 8       | 107 (0) | 53      |
| Domino| 372 (1) | 365     | 362 (1) | 125     | 366 (1) | 297     | 223 (0) | 113     |
| Med   | 1095 (3)| 1440    | 3 (0)   | 1       | 2 (0)   | 0       | 4 (0)   | 2       |
| Ring  | 1350 (3)| 1350    | 902 (2) | 640     | 901 (2) | 901     | 904 (2) | 901     |
| Sick  | 729 (2) | 729     | 1 (0)   | 0       | 1 (0)   | 1       | 64 (0)  | 63      |
| Total | 630 (13)| 673     | 166 (3) | 97      | 168 (3) | 152     | 238 (3) | 201     |

### Table A 8: Conditional planning with DEPQBQ

|       | DEPQBQ^a | DEPQBQ^b | DEPQBQ^c | DEPQBQ^d |
|-------|-----------|-----------|-----------|-----------|
| Bts1  | 722 (2)   | 720       | 364 (1)   | 361       | 396 (1)   | 366       | 721 (2) | 506     |
| Bts2  | 724 (2)   | 720       | 369 (1)   | 41       | 489 (1)   | 362       | 723 (2) | 720     |
| Bts3  | 720 (2)   | 720       | 372 (1)   | 10       | 721 (2)   | 361       | 726 (2) | 720     |
| Bts4  | 745 (2)   | 720       | 368 (1)   | 362     | 722 (2)   | 654       | 731 (2) | 720     |
| Domino| 474 (1)   | 158       | 361 (1)   | 48       | 366 (1)   | 7         | 376 (1) | 360     |
| Med   | 2 (0)     | 0         | 3 (0)     | 1       | 2 (0)     | 0         | 1 (0)   | 0       |
| Ring  | 1240 (2)  | 1350      | 901 (2)   | 900     | 902 (2)   | 901       | 902 (2) | 902     |
| Sick  | 55 (0)    | 48        | 1 (0)     | 0       | 1 (0)     | 0         | 0 (0)   | 0       |
| Total | 598 (11)  | 554       | 342 (7)   | 215     | 449 (9)   | 331       | 522 (11) | 491    |

### Table A 9: Conditional planning with QESTO

|       | QESTO^a | QESTO^b | QESTO^c | QESTO^d |
|-------|---------|---------|---------|---------|
| Bts1  | 380 (1) | 375     | 102 (0) | 65      | 369 (1)   | 42       | 400 (1) | 391     |
| Bts2  | 384 (1) | 379     | 370 (1) | 182     | 720 (2)   | 720      | 390 (1) | 378     |
| Bts3  | 390 (1) | 379     | 371 (1) | 364     | 722 (2)   | 362      | 396 (1) | 377     |
| Bts4  | 392 (1) | 382     | 379 (1) | 89      | 721 (2)   | 720      | 392 (1) | 379     |
| Domino| 720 (2) | 413     | 363 (1) | 5       | 366 (1)   | 360      | 530 (1) | 481     |
| Med   | 1100 (3)| 1440    | 3 (0)   | 1       | 1 (0)     | 0         | 5 (0)   | 6       |
| Ring  | 1350 (3)| 1350    | 901 (2) | 900     | 902 (2)   | 901       | 905 (2) | 902     |
| Sick  | 739 (2) | 736     | 1 (0)   | 1       | 1 (0)     | 0         | 11 (0)  | 11      |
| Total | 681 (14)| 681     | 311 (6) | 200     | 475 (10)  | 388       | 378 (7) | 365     |

### Table A 10: Conditional planning with QUTE

|       | QUTE^a | QUTE^b | QUTE^c | QUTE^d |
|-------|--------|--------|--------|--------|
| Bts1  | 770 (2)| 760    | 365 (1)| 361    | 724 (2) | 722    | 751 (2) | 726    |
| Bts2  | 1080 (3)| 1080  | 376 (1)| 368    | 724 (2) | 721    | 1080 (3)| 1080   |
| Bts3  | 896 (2)| 901    | 380 (1)| 370    | 783 (2) | 777    | 1080 (3)| 944    |
| Bts4  | 966 (2)| 916    | 390 (1)| 377    | 731 (2) | 726    | 1081 (3)| 1080   |
| Domino| 722 (2)| 408    | 361 (1)| 360    | 720 (2) | 720    | 606 (1) | 451    |
| Med   | 1104 (3)| 1440  | 3 (0)  | 1      | 5 (0)   | 3      | 27 (0)  | 24     |
| Ring  | 1350 (3)| 1350  | 901 (2)| 900    | 907 (2) | 903    | 917 (2) | 911    |
| Sick  | 782 (2)| 776    | 1 (0)  | 0      | 1 (0)   | 1      | 6 (0)   | 6      |
| Total | 958 (19)| 953    | 347 (7)| 342    | 574 (12)| 571    | 693 (14)| 652    |
Appendix B ASP(Q) and QASP

An ASP(Q) program $\Pi$ is an expression of the form:

$$\square_0 P_0 \square_1 P_1 \cdots \square_n P_n : C,$$

(B1)

where $n \geq 0$, $P_0, \ldots, P_n$ are logic programs, every $\square_i$ is either $\exists^* \text{ or } \forall^*$, and $C$ is a normal stratified program. In Amendola et al. 2019, the logic programs $P_i$ can be disjunctive. Here, for simplicity, we restrict ourselves to logic programs as defined in the Background section. Symbols $\exists^*$ and $\forall^*$ are called existential and universal answer set quantifiers, respectively. We say that the program $P_i$ is existentially (universally, respectively) quantified if $\square_i \text{ is } \exists^* \text{ } (\forall^*$, respectively). Let $X$ and $Y$ be sets of atoms such that $X \subseteq Y$, by $\text{fixfact}(X,Y)$ we denote the set of rules $\{x \leftarrow x \in X\} \cup \{x \leftarrow x \in Y \setminus X\}$. Let $\text{atoms}(P)$ denote the set of atoms occurring in a logic program $P$. Given an ASP(Q) program $\Pi$, a logic program $P$, and a set of atoms $X$, we denote by $\Pi_{P,X}$ the program of the form $\Pi$ such that $P_0$ is replaced by $P_0 \cup \text{fixfact}(X, \text{atoms}(P))$, that is, $\Pi_{P,X} = \square_0 (P_0 \cup \text{fixfact}(X, \text{atoms}(P))) \cdots \square_n P_n : C$. The coherence of ASP(Q) programs can be defined recursively as follows:

- $\exists^* P : C$ is coherent, if there exists $M \in SM(P)$ such that $C \cup \text{fixfact}(M, \text{atoms}(P))$ is satisfiable;
- $\forall^* P : C$ is coherent, if for every $M \in SM(P)$, $C \cup \text{fixfact}(M, \text{atoms}(P))$ is satisfiable;
- $\exists^* P \Pi$ is coherent, if there exists $M \in SM(P)$ such that $P_{M,P}$ is coherent.
- $\forall^* P \Pi$ is coherent, if for every $M \in SM(P)$, $P_{M,P}$ is coherent.

Before moving to the translations between ASP(Q) and QASP, we introduce some special forms of ASP(Q) programs. We say that an ASP(Q) program of the form (B1) is in normal form if $C$ is empty and the heads of the rules of every $P_i$ do not contain atoms occurring in any $P_j$ such that $j < i$, and we say that an ASP(Q) program is in $\forall$-GDT form if all its universally quantified logic programs are in GDT form. ASP(Q) programs of the form (B1) can be translated to ASP(Q) programs in normal and $\forall$-GDT form (using auxiliary variables) following the next steps:

1. Replace program $C$ by $\emptyset$, and add $\exists^* C$ after $\square_n P_n$.
2. In every logic program, replace the normal rules $p \leftarrow B$ such that $p$ occurs in some previous program by constraints $\perp \leftarrow \neg p, B$; and erase the choice rules $\{p\} \leftarrow B$ such that $p$ occurs in some previous program.
3. Apply the method described in Niemelä 2008 Fandinno et al. 2020 to translate the universally quantified programs to GDT form.

The translation can be computed in polynomial time on the size of the ASP(Q) program, and the result is an ASP(Q) program in normal and $\forall$-GDT form that is coherent if and only if the original ASP(Q) program is coherent.

Once an ASP(Q) program $\Pi$ of the form (B1) is in normal and $\forall$-GDT form, we can translate it to a QLP as follows. Let $\text{qlp}(\Pi)$ be the QLP of the form

$$Q_0 X_0 Q_1 X_1 \cdots Q_n X_n \{P_0^* \cup P_1^* \cup \cdots \cup P_n^* \cup O\}$$

where for every $i \in \{0, \ldots, n\}$, if $\square_i$ is $\exists^*$ then $Q_i$ is $\exists$, $X_i$ is the set of atoms occurring
in $P_i$ that do not occur in any previous program, and $P_i^\ast$ is $P_0$ if $n = 0$ and otherwise it is the program that results from adding the literal $\neg \alpha_i$ to the body of every rule of $P_i$; while if $\Box_i$ is $\forall^s$ then $Q_i$ is $\forall$, $X_i$ is $\{p \mid \{p\} \leftarrow P_i\}$, and $P_i^\ast$ is the program that results from replacing $\bot$ in the head of every integrity constraint by $\alpha_i$; and $O$ is $\{\alpha_i \leftarrow \alpha_{i-1} \mid \beta \leq i \leq n\}$ where $\beta$ is 1 if $\Box_0 = \forall^s$ and is 2 otherwise.

**Theorem Appendix B.1**

$\Pi$ is coherent if and only if $qlp(\Pi)$ is satisfiable.

The translation in the other direction is as follows. Given a QLP $QP$ over $\mathcal{A}$ of the form [2], let $aspq(QP)$ be the ASP(Q) program of the form

$$Q_0^s P_0 Q_1^s P_1 \ldots Q_n^s P_n \exists^s (P \cup O) : \emptyset$$

where every $P_i$ is $\{(p') \leftarrow p \in X_i\}$, $O$ is $\{\bot \leftarrow p, \neg p' \mid p \in \mathcal{X}\} \cup \{\bot \leftarrow \neg p, p' \mid p \in \mathcal{X}\}$, where $\mathcal{X} = \bigcup_{i \in \{0, \ldots, n\}} X_i$, and we assume that the set $\{p' \mid p \in \mathcal{X}\}$ is disjunct from $\mathcal{A}$.

**Theorem Appendix B.2**

$QP$ is satisfiable if and only if $aspq(QP)$ is coherent.
Appendix C Proof of Theorem 6.1

We first introduce some notation. Let $D$ be a planning description. By $C_D$ we denote the logic program $\text{Domain} \cup D^* \cup [3] \cup [5]$. Then (6) can be rewritten as $\forall \text{Occ}\forall \text{Open} C_D$. Let $\text{Generate}$ denote $[3]$, and $\text{Alpha}$ denote $[5]$. Then $C_D$ can be rewritten as $\text{Domain} \cup D^* \cup \text{Generate} \cup \text{Alpha}$. Since all occurrences of variable $T$ in $C_D$ are always bound by an atom $t(T)$, and the only ground atoms of that form that can be true are $t(1), \ldots, t(n)$, we can consider only the ground instantiations of $C_D$ where $T$ has the values from 1 to $n$.

We add some more notation to represent those ground instances. We denote by $r_i$ we can consider only the ground instantiations of $C_D$ where $T$ has the values from 1 to $n$. We add some more notation to represent those ground instances. We denote by $r(i)$ the rule obtained by replacing in $r$ the variable $T$ by $i$. If $i \leq j$, by $r(i, j)$ we denote the set of rules containing $r(k)$ for all $i \leq k \leq j$. For a set of rules $R$, we define $R(i) = \{r(i) \mid r \in R\}$ and $R(i, j) = \{r(i, j) \mid r \in R\}$. Let $C_D[i]$ denote the program

$$\text{Alpha}(i) \cup \text{Generate}(i) \cup D^*_G(i)$$

and $C_D[0, j]$ denote the program

$\text{Domain} \cup D^*_i \cup \bigcup_{i=1}^{j} C_D[i].$

Note that $C_D$ has the same stable models as the program

$$C_D[0, n] \cup D^*_G$$

that is the same as

$$C_D[0, j] \cup \bigcup_{i=j+1}^{n} C_D[i] \cup D^*_G$$

for every $j \in \{1, \ldots, n\}$. The following results make repetitive use of the Splitting Theorem (Lifschitz and Turner 1994). Note that no atom occurring in $C_D[0, j]$ occurs in the head of a rule in $C_D[i] \cup D^*_G$ for any $i > j$. We also define $\text{Holds}[i] = \{h(f, i) \mid f \in F\}$.

**Lemma 1**

Let $D$ be a planning description that represents a conformant planning problem $\mathcal{PP}$, $s, s' \subseteq F$ be two states, $a$ be an action, and $i$ be a positive integer. Moreover, let $X$ be a stable model of $C_D[0, i - 1]$ such that $s' = \{f \in F \mid h(f, i - 1) \in X\}$ and $\alpha(i - 1) \notin X$. Then, $\Phi(s', a) = s$ if and only if there is a unique stable model $Y$ of

$$X \cup C_D[i] \cup \text{fixcons}(\{\text{occ}(a, i)\}, \text{Occ}_i)$$

(C1)

such that $s = \{f \in F \mid h(f, i) \in Y\}$ and $\alpha(i) \notin Y$. Furthermore, if $\Phi(s', a) = \bot$ then (C1) is unsatisfiable.

**Proof**

Let

$$P_1 = X \cup \text{Alpha}(i) \cup \text{Generate}(i) \cup \text{fixcons}(\{\text{occ}(a, i)\}, \text{Occ}_i)$$

$$P_2 = P_1 \cup D^*_G(i)$$

Recall that $C_D[i] = \text{Alpha}(i) \cup \text{Generate}(i) \cup D^*_G(i)$ and, thus, $P_2$ is equal to (C1). Program $P_1$ has a unique stable model $X_1 = X \cup \{\text{occ}(a, i)\}$ and, by the Splitting Theorem, the stable models of $P_2$ are the same as the stable models of $X_1 \cup D^*_G(i)$. Note
that \( \alpha(i - 1) \notin X \) implies that \( \alpha(i) \notin X \) because the only rule with \( \alpha(i) \) in the head has \( \alpha(i - 1) \) in the body. Furthermore, since \( \mathcal{D} \) represents \( \mathcal{P}\mathcal{P} \) and \( \alpha(i) \notin X \), it follows that

\[
(X \cap \text{Holds}[i - 1]) \cup \{\text{occ}(a, i)\} \cup \{t(i)\} \cup \mathcal{D}_n^\bullet(i)
\]  

(C2)

either Case 1 has a unique stable model \( Y' \) such that \( s = \{f \in F \mid h(f, i) \in Y'\} \) or (Case 2) has no stable model and \( \Phi(s', a) = \bot \). Recall that \( \mathcal{D}(\mathcal{D}) \) is deterministic. Hence, it cannot generate multiple stable models, and the same applies to \( \mathcal{D}_n^\bullet(i) \). Note also that \( \alpha(i) \notin X \) implies \( \alpha(i) \notin Y' \) because the only rules with \( \alpha(i) \) in the head belong to \( P_1 \).

In Case 1 the iff holds because \( Y = Y' \cup X \) is the unique stable model of \( X_1 \cup \mathcal{D}_n^\bullet(i) \), and it satisfies the conditions \( s = \{f \in F \mid h(f, i) \in Y\} \) and \( \alpha(i) \notin Y \). To see this, observe that the atoms in \( X_1 \) that do not belong to (C2) also do not occur in the rules of \( \mathcal{D}_n^\bullet(i) \). In Case 2 the iff holds because none of its two sides holds. Finally, if \( \Phi(s', a) = \bot \) then (C2) is unsatisfiable, hence \( X_1 \cup \mathcal{D}_n^\bullet(i) \) is also unsatisfiable, and therefore (C1) is unsatisfiable. \( \square \)

**Lemma 2**

Let \( \mathcal{D} \) be a planning description that represents a conformant planning problem \( \mathcal{P}\mathcal{P} \), \( s, s' \subseteq F \) be two states, \( p = [a_1; a_2; \ldots; a_n] \) be a plan, \( i \) be a positive integer, and \( X \) be a stable model of \( C_D[0, i - 1] \) such that \( s' = \{f \in F \mid h(f, i - 1) \in X\} \) and \( \alpha(i - 1) \notin X \). Moreover, let

\[
A = \{\text{occ}(a_1, i), \text{occ}(a_2, i + 1), \ldots, \text{occ}(a_n, i + n - 1)\}
\]

be a set of atoms representing action occurrences. Then, \( \Phi(\{s', p\}) = s \) iff there is a unique stable model \( Y \) of

\[
X \cup \bigcup_{j=i}^{i+n-1} (C_D[j] \cup \text{fixcons}(\{\text{occ}(a, j)\}, \text{Occ}_j))
\]  

(C3)

such that \( s = \{f \in F \mid h(f, i + n - 1) \in Y\} \). Furthermore, \( \alpha(i + n - 1) \notin Y \).

**Proof**

We consider first the case where \( \Phi(s', a_1) = \bot \), and then the case where \( \Phi(s', a_1) \) is a state \( s'' \). If \( \Phi(s', a_1) \) is \( \bot \), then by Lemma 1 the program

\[
X \cup C_D[i] \cup \text{fixcons}(\{\text{occ}(a, i)\}, \text{Occ}_i)
\]  

(C4)

is unsatisfiable. Hence, by the Splitting Theorem, the program (C3) is also unsatisfiable, and the iff holds because none of its sides holds. We continue with the case where \( \Phi(s', a_1) \) is a state \( s'' \). If \( n = 1 \), the lemma statement follows directly from Lemma 1. Then the proof follows by induction assuming that the lemma statement holds for all plans of length \( n - 1 \). By definition, \( \Phi(\{s', p\}) = \Phi(\{s''\}, q) \) with \( q = [a_2; \ldots; a_n] \). From Lemma 1 we get that \( \Phi(s', a_1) = s'' \) iff there is a unique stable model \( Z \) of (C4) such that \( s'' = \{f \in F \mid h(f, i) \in Z\} \) and \( \alpha(i) \notin Z \). By induction hypothesis, we get that \( \Phi(\{s''\}, q) = s \) iff there is a unique stable model \( Y \) of

\[
Z \cup \bigcup_{j=i+1}^{i+n-1} (C_D[j] \cup \text{fixcons}(\{\text{occ}(a, j)\}, \text{Occ}_j))
\]
Then, the following two statements are equivalent:

\[ \Phi(D) = X \]

Let \( D \) be a planning description that represents a conformant planning problem \( \mathcal{PP} \). Let \( Y \subseteq \text{Open} \) be a set of atoms, \( s \subseteq F \) and \( s_0 \in SM(I(D)) \) be states such that

\[ Y = \{ h(f,0) \in \text{Open} \mid f \in s_0 \}, \]

and \( p = [a_1; a_2; \ldots; a_n] \) be a plan with

\[ X = \{ \text{occ}(a_1,1), \text{occ}(a_2,2), \ldots, \text{occ}(a_n,n) \}. \]

Then, \( \Phi(\{s_0\}, p) = s \) iff there is a unique stable model \( Z \) of

\[ C_D[0,n] \cup \text{fixcons}(X, \text{Occ}) \cup \text{fixcons}(Y, \text{Open}) \] (C5)

such that \( s = \{ f \mid h(f,n) \in Z \} \). Furthermore, \( \alpha(n) \notin Z \).

**Proof**

Note that \( C_D[0,0] = \text{Domain} \cup \mathcal{D}_1^* \) and that \( s_0 \in SM(I(D)) \) implies that this program has a unique stable model \( Z_0 \) such that \( s_0 = \{ f \in F \mid h(f,0) \in Z_0 \} \) and \( \alpha(0) \notin Z_0 \). Then, from Lemma 2 with \( i = 1 \), we get that \( \Phi(\{s_0\}, p) = s \) iff there is a unique stable model \( Z \) of

\[ Z_0 \cup \bigcup_{j=1}^{n} (C_D[j] \cup \text{fixcons}(\{\text{occ}(a,j)\}, \text{Occ})). \]

Furthermore, \( \alpha(n) \notin Z \). From the Splitting Theorem, this implies that \( Z \) is a stable model of \( C_D[0,0] \). Furthermore, it is the unique stable model of \( C_D[0,n] \) such that

\[ s_0 = \{ f \in F \mid h(f,0) \in Z_0 \} = \{ f \in F \mid h(f,0) \in Z \}. \]

Note also that \( Y = \{ h(f,0) \in \text{Open} \mid f \in s_0 \} = Z \cap \text{Open} \) and, thus, \( Z \) satisfies

\[ \text{fixcons}(Y, \text{Open}) = \text{fixcons}(Z \cap \text{Open}, \text{Open}). \]

No other stable model of \( C_D[0,0] \) satisfies these constraints. Therefore, \( Z \) is the unique stable model of \( C_D[0,0] \).

**Lemma 4**

Let \( D \) be a planning description that represents a conformant planning problem \( \mathcal{PP} \) such that \( I(D) \) is in \( \text{GDT} \) form, and \( n \) be a positive integer. Let \( p = [a_1; \ldots; a_n] \) be a plan, \( X = \{ \text{occ}(a_1,1), \ldots, \text{occ}(a_n,n) \} \) be a set of atoms, and \( Y \subseteq \text{Open} \) be another set of atoms. Then, the following two statements are equivalent:

- there is some stable model \( Z \) of \( C_D \cup \text{fixcons}(X, \text{Occ}) \cup \text{fixcons}(Y, \text{Open}) \) such that \( \alpha(n) \notin Z \); and
- there is \( s_0 \in SM(I(D)) \) s.t. \( Y = \{ h(f,0) \in \text{Open} \mid f \in s_0 \} \) and \( \Phi_D(\{s_0\}, p) \subseteq G \).
Proof
First, assume that there is no $s_0 \in SM(I(D))$ s.t. $Y = \{h(f, 0) \in Open \mid f \in s_0\}$. In this case, the second lemma statement does not hold, and we will see next that the first lemma statement does not hold either. Given our assumption and by construction, the program $Domain \cup \{I\} \cup Alpha \cup fixcons(Y, Open)$, that is a subset of $C_D \cup fixcons(X, Occ) \cup fixcons(Y, Open)$, has a unique stable model, and this stable model contains the atom $\alpha(n)$. Then, by the Splitting Theorem, the stable models of $C_D \cup fixcons(X, Occ) \cup fixcons(Y, Open)$ contain the atom $\alpha(n)$, which contradicts the first lemma statement.

Otherwise, there is $s_0 \in SM(I(D))$ such that $Y = \{h(f, 0) \in Open \mid f \in s_0\}$. Then, from Lemma 3 it follows that $\Phi_{D(D)}(\{s_0\}, p) = s$ iff there is a unique stable model $Z$ of $C_D[0, n] \cup fixcons(X, Occ) \cup fixcons(Y, Open)$ such that $s = \{f \in F \mid h(f, n) \in Z\}$. Furthermore, $\alpha(n) \notin Z$. Note that $C_D = C_D[0, n]$. Moreover, we can see that $s \in G$ iff $Z$ satisfies $D_G^c$. Since $D_G^c \subseteq D^*$, it follows that the two lemma statements are equivalent.

Lemma 5
Let $D$ be a planning description that represents a conformant planning problem $PP$ such that $I(D)$ is in GDT form. Then, the following statements hold:

- For every $Y \subseteq Open$ such that $D_I^c \cup fixcons(Y, Open)$ is satisfiable, there is a unique $s_Y \in SM(I(D))$ such that $Y = \{h(f, 0) \in Open \mid f \in s_Y\}$.
- For every $s_0 \in SM(I(D))$, there is a unique $Y \subseteq Open$ such that $Y = \{h(f, 0) \in Open \mid f \in s_0\}$. Furthermore, $D_I^c \cup fixcons(Y, Open)$ is satisfiable.

Proof
For the first statement, note that there is a choice rule of form $\{f\} \leftarrow$ in $I(D)$ iff the atom $h(f, 0)$ belongs to $Open$. The rest of the program is deterministic, therefore, there is at most one $s_0 \in SM(I(D))$ such that $Y = \{h(f, 0) \in Open \mid f \in s_0\}$. Furthermore, since $D_I^c \cup fixcons(Y, Open)$ is satisfiable, such stable model exists. For the same reason, for the second statement, if $s_0 \in SM(I(D))$, there is a unique $Y \subseteq Open$ such that $Y = \{h(f, 0) \in Open \mid f \in s_0\}$. Furthermore, it is easy to see that $D_I^c \cup fixcons(Y, Open)$ is satisfiable.

Lemma 6
Let $D$ be a planning description that represents a conformant planning problem $PP$ such that $I(D)$ is in GDT form, and $n$ be a positive integer. Let $p = [a_1; \ldots; a_n]$ be a plan, and $X = \{\text{occ}(a_1, 1), \ldots; \text{occ}(a_n, n)\}$ be a set of atoms. Then, the following two statements are equivalent:

- for all $Y \subseteq Open$ such that $D_I^c \cup fixcons(Y, Open)$ is satisfiable, there is some stable model $Z$ of $C_D \cup fixcons(X, Occ) \cup fixcons(Y, Open))$ such that $\alpha(n) \notin Z$; and
- every $s_0 \in SM(I(D))$ satisfies $\Phi_{D(D)}(\{s_0\}, p) \subseteq G$. 
Proof

From Lemma 4, the first lemma statement holds iff

for all \( Y \subseteq Open \) such that \( D^I_Y \cup fixcons(Y, Open) \) is satisfiable,
there is \( s \in SM(I(D)) \) such that \( Y = \{ h(f, 0) \in Open \mid f \in s \} \)
and \( \Phi_{D(D)}(\{s\}, p) \subseteq G \).

Furthermore, for every \( Y \subseteq Open \), there is at most one state \( s_0 \in SM(I(D)) \) such
that \( Y = \{ h(f, 0) \in Open \mid f \in s_0 \} \) (Lemma 5). Therefore, the above statement can be
rewritten as

every \( Y \subseteq Open \) satisfies that, if \( D^I_Y \cup fixcons(Y, Open) \) is satisfiable,
then \( \Phi_{D(D)}(\{s_Y\}, p) \subseteq G \) where \( s_Y \in SM(I(D)) \) is the unique state such
\[ \text{(C6)} \]
that \( Y = \{ h(f, 0) \in Open \mid f \in s_Y \} \).

First-to-second statement. Pick any \( s_0 \in SM(I(D)) \). From Lemma 5, there is a unique
set \( Y \subseteq Open \) s.t. \( Y = \{ h(f, 0) \in Open \mid f \in s_0 \} \). Furthermore, by construction, \( D^I_Y \cup fixcons(Y, Open) \) is satisfiable. From \( \text{(C6)} \), this implies \( \Phi_{D(D)}(\{s_0\}, p) \subseteq G \) and, thus, the second lemma statement holds.

Second-to-first statement. Pick any \( Y \subseteq Open \) such that \( D^I_Y \cup fixcons(Y, Open) \) is satisfiable.
From Lemma 6, this implies that there is a unique \( s_Y \in SM(I(D)) \) such
that \( Y = \{ h(f, 0) \in Open \mid f \in s_Y \} \). From the second lemma statement, this implies
that \( \Phi_{D(D)}(\{s_Y\}, p) \subseteq G \). Hence, \( \text{(C6)} \) and the first lemma statement hold.

Lemma 7

Let \( D \) be a planning description that represents a conformant planning problem \( PP \) such
that \( I(D) \) is in GDT form and \( n \) be a positive integer. Let \( p = \{a_0; \ldots; a_n\} \) be a plan,
\( X = \{ occ(a_1, 1), \ldots; occ(a_n, n) \} \) be a set of atoms, and \( Y \subseteq Open \). Then, the following
two statements are equivalent:

- \( C_D \cup fixcons(\{X, Occ\}) \cup fixcons(Y, Open) \) is satisfiable; and
- if \( D^I_Y \cup fixcons(Y, Open) \) is satisfiable, then there is some stable model \( Z \) of \( C_D \cup fixcons(\{X, Occ\}) \cup fixcons(Y, Open) \) such that \( \alpha(n) \notin Z \).

Proof

We proceed by cases. Assume first that \( D^I_Y \cup fixcons(Y, Open) \) is satisfiable. Then, the second lemma statement trivially implies the first. Next, assume that the first lemma statement holds. We show that the second lemma statement follows. By definition, the first lemma statement implies that there is some stable model \( Z \) of \( C_D \cup fixcons(\{X, Occ\}) \cup fixcons(Y, Open) \). Hence, we only need to show that \( \alpha(n) \notin Z \) and, for this, it is enough to show that \( \alpha(0) \notin Z \). Suppose, for the sake of contradiction, that \( \alpha(0) \in Z \). From the Splitting Theorem, there is a stable model \( Z_0 \) of \( D^\bullet_Y \cup fixcons(Y, Open) \) such that \( \alpha(0) \in Z_0 \). By construction, this implies that \( D^I_Y \cup fixcons(Y, Open) \) is not satisfiable, which is a contradiction with the assumption.

Assume now that \( D^I_Y \cup fixcons(Y, Open) \) is not satisfiable. Then, the second lemma statement trivially holds. Furthermore, by construction, every stable model \( Z_0 \) of \( D^\bullet_Y \cup fixcons(Y, Open) \) satisfies \( \alpha(0) \in Z_0 \). Let \( Z = Z_0 \cup X \cup \{ \alpha(i) \mid 1 \leq i \leq n \} \). It holds that

- the bodies of all rules in \( D^\bullet_Y \cup D^\bullet_G \) are not satisfied by \( Z \),
the bodies of all rules in Alpha are satisfied by Z and all α(i) are thus supported, and

all rules in Generate are satisfied by Z and all elements in X are supported.

Thus Z is a stable model of $C_D \cup \text{fixcons}(X,\text{Occ}) \cup \text{fixcons}(Y,\text{Open})$ and the first lemma statement holds. □

**Lemma 8**

Let $D$ be a planning description that represents a conformant planning problem $PP$ such that $I(D)$ is in GDT form, and $n$ be a positive integer. Let $p = [a_0; \ldots ; a_n]$ be a plan and $X = \{\text{occ}(a_0,0), \ldots \text{occ}(a_n,n)\}$ be a set of atoms. Then, the following two statements are equivalent:

- for all $Y \subseteq \text{Open}$, $C_D \cup \text{fixcons}(X,\text{Occ}) \cup \text{fixcons}(Y,\text{Open})$ is satisfiable; and
- every $s_0 \in SM(I(D))$ satisfies $\Phi_{D(D)}(\{s_0\},p) \subseteq G$.

**Proof**

The first lemma statement holds iff for all $Y \subseteq \text{Open}$ satisfying that $D \cup \text{fixcons}(Y,\text{Open})$ is satisfiable, there is some stable model $Z$ of $C_D \cup \text{fixcons}(X,\text{Occ}) \cup \text{fixcons}(Y,\text{Open})$ such that $\alpha(n) \notin Z$ (Lemma 7) iff the second lemma statement holds (Lemma 6). □

**Lemma 9**

Let $D$ be a planning description that represents a conformant planning problem $PP$ such that $I(D)$ is in GDT form, and let $n$ be a positive integer. Then, the following two statements are equivalent:

- there is some $X \subseteq \text{Occ}$ such that, for all $Y \subseteq \text{Open}$, there is some stable model of $C_D \cup \text{fixcons}(X,\text{Occ}) \cup \text{fixcons}(Y,\text{Open})$; and
- there is some $X \subseteq \text{Occ}$ of the form $\{\text{occ}(a_1,1), \ldots ,\text{occ}(a_n,n)\}$ such that, for all $Y \subseteq \text{Open}$, there is some stable model of $C_D \cup \text{fixcons}(X,\text{Occ}) \cup \text{fixcons}(Y,\text{Open})$.

**Proof**

The second statement trivially implies the first. In the other direction, assume that the first statement holds, we show that in this case the second statement also holds. Pick any $Y \subseteq \text{Open}$, and let $M$ be some stable model of $C_D \cup \text{fixcons}(X,\text{Occ}) \cup \text{fixcons}(Y,\text{Open})$. Given that the rule Generate belongs to $C_D$, it follows that $M \cap \text{Occ}$ has the form $\{\text{occ}(a_1,1), \ldots ,\text{occ}(a_n,n)\}$. On the other hand, since $M$ satisfies fixcons($X,\text{Occ}$), it follows that $M \cap \text{Occ} = X$. Hence, $X$ has the form $\{\text{occ}(a_1,1), \ldots ,\text{occ}(a_n,n)\}$, and the second statement holds. □

**Proof of Theorem 6.1**

Since $D$ represents $PP$, it follows that $PP = \langle \Phi_{D(D)}, SM(I(D)), G \rangle$ where $G$ is $SM(G(D) \cup \{\{f\} \leftarrow \emptyset | f \in F\})$. Then, the program

$$\exists \text{Occ} \cap \text{Open} C_D$$

is satisfiable iff there is some $X \subseteq \text{Occ}$ such that, for all $Y \subseteq \text{Open}$, there is some stable model of
\( C_D \cup \text{fixcons}(X, \text{Occ}) \cup \text{fixcons}(Y, \text{Open}) \)

iff there is some \( X \subseteq \text{Occ} \) of the form \( \{\text{occ}(a_1,1), \ldots, \text{occ}(a_n,n)\} \) such that, for all \( Y \subseteq \text{Open} \), there is some stable model of \( C_D \cup \text{fixcons}(X, \text{Occ}) \cup \text{fixcons}(Y, \text{Open}) \) (Lemma 9)

iff there is some \( X \subseteq \text{Occ} \) of the form \( \{\text{occ}(a_1,1), \ldots, \text{occ}(a_n,n)\} \) such that every \( s_0 \in SM(I(D)) \) satisfies \( \Phi_D(D)({s_0}, p) \subseteq G \) with \( p = [a_0; \ldots; a_n] \) (Lemma 8)

iff there is some plan \( p \) of length \( n \) s.t. every \( s_0 \in SM(I(D)) \) satisfies \( \Phi_D(D)(SM(I(D)), p) \subseteq G \)

iff there is some plan \( p \) of length \( n \) such that \( \Phi_D(D)(SM(I(D)), p) \subseteq G \)

iff there is some plan of length \( n \) that solves problem \( \mathcal{PP} \). \( \square \)