Journal of Artificial Intelligence Research 69 (2020) 923–961 Submitted 12/2019; published 11/2020

Qualitative Numerical Planning: Reductions and Complexity

Blai Bonet BONET@USB.VE
Universidad Simón Bolívar
Caracas, Venezuela

Hector Geffner HECTOR.GEFFNER@UPF.EDU
ICREA & Universitat Pompeu Fabra
Barcelona, Spain

Abstract

Qualitative numerical planning is classical planning extended with non-negative real variables that can be increased or decreased "qualitatively", i.e., by positive indeterminate amounts. While deterministic planning with numerical variables is undecidable in general, qualitative numerical planning is decidable and provides a convenient abstract model for generalized planning. The solutions to qualitative numerical problems (QNPs) were shown to correspond to the strong cyclic solutions of an associated fully observable non-deterministic (FOND) problem that terminate. This leads to a generate-and-test algorithm for solving QNPs where solutions to a FOND problem are generated one by one and tested for termination. The computational shortcomings of this approach for solving QNPs, however, are that it is not simple to amend FOND planners to generate all solutions, and that the number of solutions to check can be doubly exponential in the number of variables. In this work we address these limitations while providing additional insights on QNPs. More precisely, we introduce two polynomial-time reductions, one from QNPs to FOND problems and the other from FOND problems to QNPs both of which do not involve termination tests. A result of these reductions is that QNPs are shown to have the same expressive power and the same complexity as FOND problems.

1. Introduction

Qualitative numerical problems (QNPs) are classical planning problems extended with non-negative numerical variables X that can be decreased or increased "qualitatively", i.e., by positive indeterminate amounts. Since such numerical variables cannot be used for counting, QNP planning, unlike most other general forms of planning with numbers (Helmert, 2002), turns out to be decidable. QNPs were introduced by Srivastava et al. (2011) as a useful model for *generalized planning*, namely, the synthesis of plans that solve multiple classical planning instances (Levesque, 2005; Bonet et al., 2009; Srivastava et al., 2011; Hu & De Giacomo, 2011; Bonet & Geffner, 2015; Belle & Levesque, 2016; Jiménez-Celorrio et al., 2019). Basically, collections $\mathcal{Q}$ of planning instances P that share the same set of actions and state features may be often expressed as a single QNP problem Q whose solutions, that map state features into actions, solve all problems P in $\mathcal{Q}$ (Bonet & Geffner, 2018).

QNPs can be solved in two steps (Srivastava et al., 2011). First, the QNP Q is converted into a standard fully observable non-deterministic (FOND) problem P (Cimatti et al., 2003). Then, solutions of P obtained by an off-the-shelf FOND planner are tested for *termination*. This last step is required because the non-determinism in the FOND problem P is not fair in the usual sense: the trajectories that are not fair are those in which a variable is decreased an infinite number of times but increased a finite number of times only (Bonet et al., 2017). The policies that solve P in which the (QNP) fair trajectories all reach the goal, and which correspond to the terminating strong cyclic policies for P, are the policies that solve the QNP Q (Srivastava et al., 2011).

The computational shortcomings of solving QNPs following this generate-and-test approach, however, are two. First, it is not simple to amend FOND planners to generate all the solutions of a FOND problem because FOND plans are not action sequences but closed-loop policies. Second, the number of policies that need to be tested for termination may be huge: exponential in the number of FOND states, and hence, doubly exponential in the number of variables.

In this work we address these limitations while providing additional insights on QNPs. We introduce two polynomial-time reductions, one from QNPs to FOND problems and the other from FOND problems to QNPs. As every (formal) reduction, both reduction are sound and complete, and hence do not require termination tests. A result of these reductions is that QNPs are shown to have the same expressive power and in particular, the plan-existence decision problem for both have the same complexity EXP-Complete (Littman et al., 1998; Rintanen, 2004). The new QNP to FOND translator is implemented and available. In combination with sound and complete FOND planners, the translation yields the only sound and complete QNP planner available (i.e., solver that works directly on the factored QNP representation without explicitly flattening the input QNP).

The structure of the paper is as follows. We review first classical planning, FOND planning, QNPs, the direct translation of QNPs into FOND problems, and the termination test. We follow ideas from Srivastava et al. (2011) but in a slightly different and more expressive formulation. We then introduce the two new reductions: from FOND problems into QNPs, and from QNPs into FOND problems. This last reduction is very different from the one sketched by Bonet et al. (2017) which is incorrect (we illustrate this with an example). We then consider variations and extensions of QNPs, the use of QNPs for generalized planning, experimental results, and related work.

2. Classical and FOND planning

A classical planning problem is a sequential decision problem where a goal is to be reached by performing actions with deterministic effects from a given initial state. These problems are usually expressed in compact form in planning languages such as STRIPS (Fikes & Nilsson, 1971; Russell & Norvig, 2002). A (grounded) STRIPS planning problem (with negation) is a tuple $P = \langle F, I, O, G\rangle$ where F denotes a set of propositional variables, I and G are sets of F-literals representing the initial and goal situation, and O is a set of actions a with preconditions and effects $Pre(a)$ and $\mathit{Eff}(a)$ given by sets of F-literals.

The *state model* $\mathcal{S}(P)$ for the problem $P = \langle F, I, O, G\rangle$ is a tuple $\mathcal{S}(P) = \langle S, s_0, Act, A, f, S_G\rangle$ where S is the set of possible truth-valuations over the F literals, called the states, s_0 is the initial state, $Act = O$, $A(s)$ represents the actions a in Act whose preconditions are true in s, $f(a, s)$ represents the state s' that follows action a in s for $a \in A(s)$, and S_G is the set of goal states. It is assumed that the problem P is consistent in the sense that s_0 and f are well-defined and S_G is not empty. A solution to a classical problem P is an action sequence $a_0, \ldots, a_n$ that generates a state sequence $s_0, \ldots, s_{n+1}$ over the model $\mathcal{S}(P)$ that reaches the goal. In this sequence, $a_i \in A(s_i)$ and $s_{i+1} = f(a_i, s_i)$ for $i = 0, \ldots, n$, and $s_{n+1} \in S_G$.

A *fully-observable non-deterministic (FOND) problem* P is like a classical planning problem except that actions a may have non-deterministic effects expressed as $\mathit{Eff}_1(a) \,|\, \cdots \,|\, \mathit{Eff}_n(a)$ where $\mathit{Eff}_i(a)$ is a set of F-literals as above (Cimatti et al., 2003; Geffner & Bonet, 2013; Ghallab et al., 2016). The state model $\mathcal{S}(P)$ determined by a FOND problem $P = \langle F, I, O, G\rangle$ is a tuple $\mathcal{S}(P) = \langle S, s_0, Act, A, F, S_G\rangle$ as above with the difference that the state transition function F is non-deterministic, and maps an action a and state s into a non-empty set $F(a, s)$ of possible successor states. As usual, the non-deterministic transition function F is given in factored form.

That is, for action a made of multiple effects $\mathit{Eff}_1 \mid \cdots \mid \mathit{Eff}_n$ (possibly deterministic when $n = 1$), each outcome s' in $F(a, s)$ results of the choice of one Eff_i for each non-deterministic effect of a.[1]

The solutions of FOND problems ensure that the goal is reached with certainty under certain fairness assumptions. Policies or plans in the FOND setting are partial functions π mapping states s into actions $\pi(s)$. A state trajectory $s_0, s_1, \ldots, s_n$ (finite or infinite) is induced by π over the model $\mathcal{S}(P)$ if the action $a_i = \pi(s_i)$ is defined, it is applicable in the state s_i, i.e., $a_i \in A(s_i)$, and s_{i+1} is in $F(a_i, s_i)$, $i = 1, \ldots, n-1$. The trajectory is said to be a *π-trajectory.* The trajectory is *maximal* if A) it is infinite, i.e., $n = \infty$, and does not include a goal state, B) if s_n is the first goal state in the sequence, or C) the action $\pi(s_n)$ is not defined or not applicable in s_n.

A policy π is a solution of the FOND problem P if all the *fair* maximal trajectories induced by π over the model $\mathcal{S}(P)$ are goal reaching (Cimatti et al., 1998, 2003). The so-called *strong solutions* assume that all state trajectories are fair. *Strong-cyclic solutions*, on the other hand, assume that all trajectories are fair *except* the infinite trajectories where a state s occurs infinitely often but a state transition (s, s') for some $s' \in F(a, s)$ for $a = \pi(s)$, occurs finitely often. The latter trajectories are deemed to be *unfair*.

Other *equivalent* characterizations of strong and strong cyclic solutions are common. For example, a strong cyclic solution π for a FOND problem P is also a policy π such that for each π-trajectory connecting an initial state to a state s, there is a π-trajectory connecting s to a goal state. Similarly, a strong solution is a strong cyclic solution π with no cycles; i.e., one where no π-trajectory visits the same state twice.

Strong solutions can also be thought as winning strategies against an adversary, while strong cyclic solutions as winning strategies against nature. Indeed, there is a well known relation between (proper) policies that achieve the goal with probability 1 in goal-based MDPs (Markov Decision Processes) and the strong cyclic policies that solve the FOND problem associated with the MDP, where the transition function is such that $F(a, s)$ collects the states s' that are possible after action a in s, i.e., for which $P_a(s'|s) > 0$ (Geffner & Bonet, 2013).

From now, by solution of a FOND problem we mean a *strong cyclic solution* or *plan*, and by a FOND planner, we mean a strong cyclic planner, i.e., a planner that produces strong cyclic solutions. There are some good FOND planners available, including PRP (Muise et al., 2012), based on classical planners, MyND (Bercher & Mattmüller, 2009), based on heuristic AND/OR search, and FOND-SAT (Geffner & Geffner, 2018), based on a reduction to SAT.

3. Qualitative Numerical Problems

Qualitative numerical problems (QNPs) are classical planning problems extended with numerical variables that can be decremented or incremented "qualitatively". We make this formal below.

3.1 QNPs: Syntax

The syntax of QNPs is defined as an extension of the STRIPS language with negation. A QNP is a tuple $Q = \langle F, V, I, O, G \rangle$ where the new component is a set V of *non-negative numerical variables* $X \in V$. These variables introduce the non-propositional atoms $X = 0$ and their negations, denoted as $X > 0$. These literals can appear in the initial situation, action preconditions, and goals of Q. The effects of actions a on a numerical variable X can be only qualitative increments or qualitative

1. As it is standard, any choice of effects is assumed to be consistent (i.e., any pair of choices for two different non-deterministic effects of the *same action* contain no complementary literals). However, with some (polynomially bounded) extra work, our methods, algorithms and results still apply if the model is extended with *constraints* that every outcome s' must satisfy, when such constraints are given in suitable form; e.g. DNF formulas over F.

decrements denoted by the expressions $Inc(X)$ and $Dec(X)$, often abbreviated as $X\uparrow$ and $X\downarrow$ respectively. We refer to $X = 0$ and $X > 0$ as the V-literals for $X \in V$, and to p and $\neg p$ for $p \in F$, as the F-literals in Q.

Definition 1. *A QNP is a tuple $Q = \langle F, V, I, O, G\rangle$ where F and V are sets of propositional and numerical variables respectively, I and G denote the initial and goal situations, and O is a set of actions a with preconditions, and propositional and numerical effects that are denoted as $Pre(a)$, $Eff(a)$, and $N(a)$ respectively. The F-literals can appear in I, G, $Pre(a)$, and $Eff(a)$, while V-literals can appear in I, G, and $Pre(a)$. The numerical effects $N(a)$ only contain special atoms of the form $Inc(X)$ or $Dec(X)$ for the variables X in V. Actions with the $Dec(X)$ effect must feature the precondition $X > 0$ for any variable X in V.*

The preconditions and effects of an action a are denoted as pairs $\langle Pre(a); Eff(a), N(a)\rangle$ where $N(a)$ contains the numerical effects; namely, expressions like $X\uparrow$ and $X\downarrow$ that stand for the increment and decrement of variable X respectively. QNPs are assumed to be syntactically consistent by requiring that no pair of complementary literals or qualitative effects appears in the initial situation, action effects, or goals. A pair of complementary literals or qualitative effects has the form $\{p, \neg p\}$ for some p in F, or $\{X = 0, X > 0\}$ or $\{X\downarrow, X\uparrow\}$ for some X in V.

Example. An *abstraction* that is suitable for expressing the generalized problem of achieving the goal $clear(x)$ over an arbitrary Blocksworld instance (Bonet & Geffner, 2018) is given in terms of the QNP $Q_{clear} = \langle F, V, I, O, G\rangle$ where $F = \{H\}$ contains a boolean variable H that represents if the gripper is holding a block, $V = \{n(x)\}$ contains a numerical variable $n(x)$ that represents the number of blocks above x, and $I = \{\neg H, n(x) > 0\}$ and $G = \{n(x) = 0\}$ represent the initial and goal situations. The actions $O = \{a, b\}$ are

$$a \;=\; \langle \neg H, n(x) > 0; H, n(x)\downarrow\rangle \tag{1}$$

and

$$b \;=\; \langle H; \neg H\rangle\,. \tag{2}$$

It is easy to see that the first action a picks up blocks that are above x; its first precondition $\neg H$ expresses that the gripper is holding no block, while the second $n(x) > 0$ that there is at least one block above x. The effects, on the other hand, make H true (expressing that some block is being held) and decrease the number $n(x)$ of blocks above x. The other action b puts the block being held away from block x as expressed by the precondition H and effect $\neg H$. The fact that b puts blocks away from block x is reflected in that it does not affect the variable $n(x)$.

The QNP Q_{clear} captures the relevant part of the infinite collection of Blocksworld instances where the goal is to achieve the atom $clear(x)$ for some block x. The solution to Q_{clear} provides the general strategy for solving all such instances. For ways of learning such abstractions automatically; see the recent work of Bonet et al. (2019). □

3.2 QNPs: Semantics

A state s for QNP $Q = \langle F, V, I, O, G\rangle$ is a valuation that assigns a truth value $s[p]$ to each boolean variable $p \in F$, and a non-negative real value $s[X]$ to each numerical variable $X \in V$. Since the initial situation I can only feature atoms of the form $X = 0$ or $X > 0$, there is a *set* S_0 of possible initial states s_0 that correspond to the valuations that satisfy the literals in I. For example, in Q_{clear}, I is given by the literals $I = \{\neg H, n(x) > 0\}$, meaning that S_0 contains all and only the

valuations that make H false and $n(x) = r$ for some positive real number r. The use of variables that can take real values for representing integer counters illustrates that the semantics of QNPs is coarse-grained, and for this reason, decidable. Indeed, QNPs use just one qualitative property of numbers; namely, that a non-negative variable eventually must reach the value of zero if it keeps being decremented and not incremented. This property is true for integers, and it also true for reals, as long as the magnitude of the decrements is bounded from below by some positive ϵ-parameter. More about this below. The state model $\mathcal{S}(Q)$ represented by a QNP can be characterized as follows:

Definition 2. *A QNP $Q = \langle F, V, I, O, G\rangle$ determines a non-deterministic state model $\mathcal{S}(Q) = \langle S, S_0, Act, A, F, S_G\rangle$ where*

- *the states s in S are the valuations that assign a truth value to the boolean variables in F and a non-negative real value to the numerical variables in V,*
- *the initial states s_0 in S_0 are those that satisfy the literals in I under a closed-world assumption (s_0 makes p and $X = 0$ false if the literals p and $X = 0$ are not in I),*
- *the actions in Act are those in O; i.e., $Act = O$,*
- *the actions $A(s)$ applicable in s are those in Act such that $Pre(a)$ is true in s,*
- *the goal states in S_G are those that satisfy G,*
- *the transition function F is such that $s' \in F(a, s)$ for $a \in A(s)$ if*
  - *(a) $s'[p]$ is true (resp. false) if p (resp. $\neg p$) is in $Eff(a)$,*
  - *(b) $s[X] < s'[X]$ if $Inc(X)$ is in $N(a)$,*
  - *(c) $s'[X] < s[X]$ if $Dec(X)$ is in $N(a)$,*
  - *(d) $s'[p] = s[p]$ if neither p nor $\neg p$ in $Eff(a)$,*
  - *(e) $s'[X] = s[X]$ if neither $X\uparrow$ nor $X\downarrow$ in $N(a)$.*

A *trajectory* $s_0, a_0, s_1, a_1, \ldots, s_n$ is compatible with the model $\mathcal{S}(Q) = \langle S, S_0, Act, A, F, S_G\rangle$ if $s_0 \in S_0$, and $a_i \in A(s_i)$ and $s_{i+1} \in F(a_i, s_i)$ for each a_i in the sequence. The trajectory is an ϵ-bounded trajectory or *ϵ-trajectory* if the numerical changes are bounded from below by a parameter $\epsilon > 0$, except when this would make the variable negative:

Definition 3. *A trajectory $s_0, a_0, s_1, a_1, \ldots, s_n$ is an* ϵ-trajectory *iff for any variable X and time point i, with $i < n$, $s_{i+1}[X] \neq s_i[X]$ implies $|s_{i+1}[X] - s_i[X]| \geq \epsilon$ or $s_{i+1}[X] = 0$.*

Trajectories bounded by $\epsilon > 0$ cannot decrease the value of a variable asymptotically without ever reaching the value of zero. This is in agreement with the key assumption in QNPs by which variables that keep being decreased and not increased eventually must reach the value zero. From now, **trajectories over QNPs will refer to ϵ-trajectories for some $\epsilon > 0$.**

3.3 QNPs: Solutions

Solutions to QNPs take the form of partial functions or policies π that map states into actions. The choice of the action $\pi(s)$ to be done in a state s, however, can only depend on the truth values $s[p]$ associated with the boolean variables p in F and the truth values of the expressions $s[X] = 0$ associated with the numerical variables X in V. If we use the notation $s[X = 0]$ to refer to $s[X] = 0$,

then $\pi(s)$ must depend solely on the *truth-valuation over the F-literals p and the V-literals* $X = 0$ that are determined by the state s. There is indeed a finite number of such truth valuations but an infinite number of states. We refer to such truth valuations as the *boolean states* of the QNP and denote the boolean state associated with a state s as $\bar{s}$.

Definition 4 (Policy)**.** *A policy* π *for a QNP* $Q = \langle F, V, I, O, G\rangle$ *is a partial mapping of states into actions such that* $\pi(s) = \pi(s')$ *if* $\bar{s} = \bar{s}'$*.*

A trajectory $s_0, a_0, s_1, a_1, \ldots, s_n$ compatible with the model $\mathcal{S}(Q)$ is said to be a π*-trajectory* for Q if $a_i = \pi(s_i)$. Sometimes, a π-trajectory is simply denoted as a sequence of states since the actions are determined by π. A π-trajectory is also said to be a *trajectory induced by* π or compatible with π. As before, a π-trajectory is *maximal* if A) the trajectory is infinite and does not include a goal state, B) s_n is the first goal state in the trajectory, or C) $\pi(s_n)$ is undefined or denotes an action that is not applicable in s_n. The solutions to QNPs are defined then as follows:

Definition 5 (Solution)**.** *Let* Q *be a QNP and let* π *be a policy for* Q*. The policy* π solves Q *iff for every* $\epsilon > 0$*, all the maximal* ϵ*-trajectories induced by* π *reach a goal state.*

We will see that solutions to QNPs can be characterized equivalently in terms of a suitable notion of *QNP-fairness*; namely, a policy π solves Q iff every maximal (QNP) fair trajectory induced by π reaches the goal, where the *unfair* trajectories are those in which some variable X is decreased infinitely often but increased finitely often. For historical reasons, such unfair trajectories are called *terminating* instead, as indeed they cannot go on forever if the decrements are bounded from below by some $\epsilon > 0$.

Example. Consider the QNP $Q_{clear} = \langle F, V, I, O, G\rangle$ from above with $F = \{H\}$, $V = \{n(x)\}$, $I = \{\neg H, n(x) > 0\}$, $G = \{n(x) = 0\}$, and $O = \{a, b\}$ where $a = \langle \neg H, n(x) > 0; H, n(x)\downarrow\rangle$ and $b = \langle H; \neg H\rangle$. Let π be the policy defined by the rules:

$$\text{if } \neg H \text{ and } n(x) > 0, \text{ then do } a\,, \tag{3}$$

$$\text{if } H \text{ and } n(x) > 0, \text{ then do } b\,. \tag{4}$$

All the maximal ϵ-bounded trajectories that are induced by the policy π on Q_{clear} have the form

$$s_0, a, s_1, b, s_2, a, s_3, b, \ldots, s_{2m}, a, s_{2m+1} \tag{5}$$

where s_{m+1}, for positive integer m, is the first state where $n(x) = 0$ is true. The actions a and b alternate because the first makes H false and the second makes it true. In each transition (s_i, s_{i+1}) for a non-negative even integer i, the numerical variable $n(x)$ decreases by ϵ or more, unless $s_{i+1}[n(x)] = 0$. The former case cannot happen more than $s_0[n(x)]/2\epsilon$ times, as the numerical variable $n(x)$ is decreased every two steps and is never increased. Thus, in all cases and for any $\epsilon > 0$, any ϵ-trajectory induced by the policy π reaches a goal state in a finite number of steps, regardless of the initial value $s_0[n(x)]$ of $n(x)$, and regardless of the actual magnitude of the changes $|s_{i+1}[n(x)] - s_i[n(x)]|$. □

Example. A more interesting QNP that requires "nested loops" is $Q_{nest} = \langle F, V, I, O, G\rangle$ with $F = \emptyset$, $V = \{X, Y\}$, $I = \{X > 0, Y > 0\}$, $G = \{X = 0\}$, and $O = \{a, b\}$ where

$$a \;=\; \langle X > 0, Y = 0; X\downarrow, Y\uparrow\rangle\,, \tag{6}$$

$$b \;=\; \langle Y > 0; Y\downarrow\rangle\,. \tag{7}$$

The policy π is given by the rules:

$$\text{if } X > 0 \text{ and } Y = 0, \text{ then do } a\,, \tag{8}$$

$$\text{if } X > 0 \text{ and } Y > 0, \text{ then do } b\,. \tag{9}$$

The policy decrements Y using action b until the action a that decreases X and increases Y can be applied, and the process is repeated until $X = 0$. The ϵ-trajectories induced by π have the form

$$s_0, b, \ldots, b, s^1_{k_0}, \quad a, s^1_0, b, \ldots, b, s^1_{k_1}, \quad a, s^2_0, b, \ldots, b, s^2_{k_2}, \quad \ldots, \quad a, s^m_0, b, \ldots, b, s^m_{k_m}, \quad a, s_G\,. \tag{10}$$

where there is an outer loop that is executed a number of times m bounded by $s_0[X]/\epsilon$, as X is decreased by ϵ or more, but is not increased. In the iteration i of such a loop, the action b is executed a number of times k_i bounded by $s^i_0[Y]/\epsilon$ as in such inner loop Y begins with value $s^i_0[Y]$ and it is decreased and not increased. The result is that all the ϵ-trajectories induced by π reach a goal state in a finite number of steps that cannot be bounded a priori because the increments of Y produced by the action a are finite but not bounded. The policy π thus solves Q_{nest}. □

4. Direct Translation and Termination Test

The problem of deciding the existence of a policy that solves a given QNPs is decidable as noted by Srivastava et al. (2011). They hint a generate-and-test procedure to find such a policy where the QNP is first translated into a FOND problem, and then all the possible strong cyclic policies for the FOND problem are enumerated and tested for termination. The translation runs in polynomial (linear) time in the number of boolean states for the QNP while the termination test for a given strong cyclic solution is polynomial in the number of FOND states. However, the number of strong cyclic solutions that need to be tested is exponential in the number of FOND states in the worst case. The generate-and-test approach is not efficient but it is complete and runs in finite time. In contrast, the plan-existence problem for *numerical planning is undecidable* even in the classical setting where there is a single initial state and the action effects are deterministic; e.g., Post's correspondence problem can be reduced to a numerical planning problem (Helmert, 2002). The decidability of plan existence for QNPs is due to the "qualitative" behaviour of the numerical variables that cannot keep track of counts; in particular, the variables cannot be incremented or decremented by specific amounts nor queried about specific values. We review the translation and the termination test for QNPs before considering a novel polynomial translation which does not require termination tests and which thus is a true reduction of QNPs into FOND problems.

The translation T_D from a QNP Q to a FOND $P = T_D(Q)$ by Srivastava et al. (2011) is simple and direct, and it involves three steps: 1) the literals $X = 0$ and $X > 0$ are made propositional with the numerical variables X eliminated, 2) $Inc(X)$ effects are converted into deterministic boolean effects $X > 0$, and 3) $Dec(X)$ effects are converted into *non-deterministic* boolean effects $X > 0 \,|\, X = 0$.

Definition 6 (Direct Translation T_D)**.** *For QNP $Q = \langle F, V, I, O, G\rangle$, the FOND problem $P = T_D(Q)$ is $P = \langle F', I', O', G'\rangle$ with*

1. *$F' = F \cup \{X = 0 : X \in V\}$, where $X = 0$ stands for a new propositional symbol $p_{X=0}$ and $X > 0$ stands for $\neg p_{X=0}$,*
2. *$I' = I$ but with $X = 0$ and $X > 0$ denoting $p_{X=0}$ and $\neg p_{X=0}$,*
3. *$O' = O$ but with $Inc(X)$ effects replaced by the deterministic propositional effects $X > 0$, and $Dec(X)$ effects replaced by non-deterministic propositional effects $X > 0 \,|\, X = 0$,*

4. $G' = G$ *but with* $X = 0$ *and* $X > 0$ *denoting* $p_{X=0}$ *and* $\neg p_{X=0}$.

The problem $P = T_D(Q)$ is a special type of FOND problem. For example, from its definition, there is no action in P that can achieve a proposition $X = 0$ deterministically. We refer to actions in the FOND P with effects $X > 0$ and $X > 0 \,|\, X = 0$ as $Inc(X)$ and $Dec(X)$ actions, as such effects in P may only come from $Inc(X)$ and $Dec(X)$ effects in Q. Also, observe that the FOND P has a unique initial state even though the QNP Q may have an infinite number of initial states.

The states of the FOND problem $P = T_D(Q)$ are related to the *boolean states* over Q, i.e., the truth-assignments over the atoms p and $X = 0$, the latter of which stand for (abbreviation of) symbols in P. A policy π for the QNP Q thus induces a policy over the FOND problem P and vice versa.[2] Moreover, the FOND problem $P = T_D(Q)$ captures the *possible boolean state transitions* in Q exactly. More precisely, (s, a, s') is a possible transition in Q iff $(\bar{s}, a, \bar{s}')$ is a possible transition in P. Indeed, if we extend the notion of strong cyclic policies to QNPs:

Definition 7. *Let Q be a QNP and let π be a policy for Q. π is strong cyclic for Q iff for every π-trajectory connecting s_0 with a state s, there is a π-trajectory connecting s with a goal state.*

The following correspondence between boolean states in Q and the states of the boolean FOND problem $T_D(Q)$ results:

Theorem 8. *Let Q be a QNP and let π be a policy for Q. π is strong cyclic solution for Q iff π is strong cyclic policy for the FOND problem $T_D(Q)$.*

Proof. Let $\mathcal{S}(Q) = \langle S, S_0, Act, A, F, S_G \rangle$ and $\mathcal{S}(P) = \langle S', s'_0, Act', A', F', S'_G \rangle$ be the state models for the QNP Q and the FOND problem $P = T_D(Q)$. From the definition of the T_D translation, the state s is in S_0 (resp. in S_G) iff $\bar{s} = s'_0$ (resp. in S'_G), and the state $s' \in F(a, s)$ for $a \in A(s)$ iff $\bar{s}' \in F'(a, \bar{s})$ for $a \in A'(\bar{s})$. This means that there is a π-trajectory connecting an initial state s_0 in S_0 with a state s in S iff there is a corresponding π-trajectory connecting s'_0 with $\bar{s}$ in S', and similarly, there is a π-trajectory connecting s with a goal state s' iff there is a corresponding π-trajectory connecting $\bar{s}$ with $\bar{s}'$ in $\mathcal{S}(Q)$. □

The correspondence between the π-trajectories connecting states s in Q and the π-trajectories connecting the states $\bar{s}$ in $P = T_D(Q)$ does not imply however that the solutions of P and Q are the same. Indeed, the $Dec(x)$ effects of an action a in Q are mapped into the non-deterministic propositional effects $X > 0 \,|\, X = 0$ in $P = T_D(Q)$ which implies that $X = 0$ will be true in P if the action a is repeated infinitely often. On the other hand, a $Dec(X)$ effect in Q ensures that $X = 0$ will be true if a is repeated infinitely often *as long as no* $Inc(X)$ *action is performed infinitely often as well.*

In other words, the correspondence between the state transitions (s, a, s') in Q and the state transitions $(\bar{s}, a, \bar{s}')$ in $P = T_D(Q)$ does not extend to *infinite trajectories* (Bonet et al., 2017). Recall that trajectories in Q refer to ϵ-trajectories for some $\epsilon > 0$ that exclude "infinitesimal" changes. As a result:

Theorem 9. *Let Q be a QNP and let π be a policy for Q. If $\tau = s_0, s_1 \ldots$ is an infinite π-trajectory in Q, then $\bar{\tau} = \bar{s}_0, \bar{s}_1, \ldots$ is an infinite π-trajectory in $P = T_D(Q)$. Yet, there may be infinite π-trajectories in $P = T(D)$ that do not correspond to any π-trajectory in Q.*

2. The policy π over states s of Q determines the policy π' over the FOND P where $\pi'(t) = \pi(s)$ if $t = \bar{s}$, and vice versa, a policy π' for P determines a policy π for Q where $\pi(s) = \pi'(t)$ if $\bar{s} = t$. For simplicity, we use the same notation π to refer to the policy π over Q and the policy π' that it induces over $P = T_D(Q)$.

Proof. For the first part, if τ is an infinite π-trajectory over Q, then $s_{i+1} \in F(a_i, s_i)$ for $a_i = \pi(s_i)$; therefore $\bar{s}_{i+1} \in F(a_i, \bar{s}_i)$ for $a_i = \pi(\bar{s}_i)$, and hence $\bar{\tau} = \bar{s}_0, \bar{s}_1, \ldots$ is an infinite π-trajectory over P.

For the second part, one example suffices. Let Q be a QNP with a single variable X that is numerical, a single action a with precondition $X > 0$ and effect $Dec(X)$, initial condition $X > 0$, and goal $X = 0$. In the state model $\mathcal{S}(P)$ associated with the FOND problem $P = T_D(Q)$, there are two states t and t', the first where $X = 0$ is true and the second where $X > 0$ is true, and there is an infinite trajectory $\bar{s}_0, \bar{s}_1, \ldots$ where all $\bar{s}_i = t'$ and $\pi(\bar{s}_i) = a$, but there is no infinite trajectory $s_0, s_1 \ldots$ in Q where $X > 0$ stays true forever while being decremented. Indeed, for any $\epsilon > 0$ and any initial value of X, $s_0[X] > 0$, it is the case that $s_n[X] = 0$ for $n > s_0[X]/\epsilon$. □

The notion of *termination* is aimed at capturing the infinite π-trajectories over the FOND problem $P = T_D(Q)$ that do not map into infinite π-trajectories over Q. Let

$$\bar{s}_0, \bar{s}_1, \ldots, [\bar{s}_i, \ldots, \bar{s}_m]^* \tag{11}$$

denote *any infinite* π-trajectory on the FOND P where the states $\bar{s}_i, \ldots, \bar{s}_m$ in brackets make the non-empty set of *recurring states*; namely those that occur infinitely often in the trajectory (not necessarily in that order). We refer to such set of recurrent states as the *loop* of the trajectory. Observe that knowledge of the loop (and the policy π) is sufficient to infer whether a variable X is decremented or incremented infinitely often. Termination imposes the following condition on loops:

Definition 10 (Terminating Trajectories)**.** *Let Q be a QNP and let π be a policy for Q. An infinite π-trajectory $\bar{s}_0, \ldots, [\bar{s}_i, \ldots, \bar{s}_m]^*$ is* terminating *in $P = T_D(Q)$ if there is a variable X in Q that is decremented but not incremented in the loop; i.e., if $\pi(\bar{s}_k)$ is a $Dec(X)$ action for some $k \in [i, m]$, and $\pi(\bar{s}_j)$ is not an $Inc(X)$ action for any $k \in [i, m]$.*

The notion of termination is a notion of fairness that is different from the one underlying strong cyclic planning that says that infinite but terminating trajectories in P are not "fair" and hence can be ignored. Indeed, this notion of termination closes the gap in Theorem 9:

Theorem 11. *Let Q be a QNP and let π be a policy for Q. $\bar{\tau} = \bar{s}_0, \bar{s}_1, \ldots$ is an infinite* non-terminating *π-trajectory in $P = T_D(Q)$ iff there is an infinite π-trajectory $\tau = s_0, s_1, \ldots$ in Q.*

Proof. Let $\tau = s_0, s_1, \ldots$ be an infinite π-trajectory in Q, and let us assume that the infinite trajectory $\bar{\tau} = \bar{s}_0, \bar{s}_1, \ldots$ is terminating. Then there must be a variable X that is decremented by $\pi(s)$ in some recurring state s in τ and which is not incremented by $\pi(s')$ on any recurrent state s' in τ. Let $s(t)$ denote the state at time point t in τ, let t be the last time point where variable X is increased in τ ($t = -1$ if X is not increased in τ), and let $X(t+1)$ be the value of variable X at the next time point. The maximum number of times that X can be decreased after $t + 1$ is bounded by $X(t+1)/\epsilon$, and after this, X must have zero value. But in τ, X is decreased an infinite number of times, in contradiction with the assumption that any action that decrements X features $X > 0$ as precondition.

For the converse, we show that one such trajectory τ in Q can be constructed for any $\epsilon > 0$, given that the trajectory $\bar{\tau}$ in P is non-terminating. We do so by adjusting the non-deterministic increments and decrements of the actions, all of which have to be greater than or equal to ϵ, except when this would result in negative values that are increased back to zero. We construct $\tau = s_0, s_1, \ldots$ from $\bar{\tau} = \bar{s}_0, \bar{s}_1, \ldots$ as follows. The value of the boolean variables is the same in s_i as in $\bar{s}_i$, and in addition, $s_i[X] = 0$ iff $X = 0$ is true in $\bar{s}_i$ for all i. We just have to find exact values for the numerical variables X in each of the states s_i in τ, and this is a function of their initial values $s_0[X]$ when $s_0[X] > 0$, and the positive decrements or increments $\Delta(X, s_i)$ when $\pi(s_i)$ is a

$Dec(X)$ or $Inc(X)$ action, and $\Delta(X, s_i) \geq \epsilon$. For simplicity and without loss of generality, let us assume that $\epsilon < 1$ (the case for $\epsilon \geq 1$ is an easy exercise).

All the positive initial values of numerical variables, increments, and decrements are set to *positive integers* by considering the sequence of actions $\pi(s_i)$, $i = 1, \ldots$. The initial values $s_0[X]$ are set to $1 + k(X, 0)$ where $k(X, i)$ stands for the number of $Dec(X)$ actions that occur between the state s_i and the first state s_j after s_i where an $Inc(X)$ action occurs (if no $Inc(X)$ action occurs after state s_i, $k(X, i)$ is the number of $Dec(X)$ actions after s_i). That is, $k(X, i)$ is the cardinality of the set

$$\{j : i \leq j < ind(X, i) \text{ and } \pi(s_j) \text{ is } Dec(X) \text{ action}\} \tag{12}$$

where $ind(X, i)$ is the minimum index $j > i$ such that $\pi(s_j)$ is an $Inc(X)$ action, or ∞ if there is no such action after s_i. Observe that $k(X, i)$ is bounded. The only way it could be infinite is when no $Inc(X)$ action occurs after s_i while at the same time an infinite number of $Dec(X)$ actions occur; yet, this is impossible since then X eventually becomes zero after which no $Dec(X)$ action may occurs as such actions feature the precondition $X > 0$. Likewise, the increments $\Delta(X, s_i)$ are set to $1 + k(X, i)$, and the decrements $\Delta(X, s_i)$ are set to $s_i[X]$ if $X = 0$ is true in $\bar{s}_{i+1}$ and to 1 is $X > 0$ if true in $\bar{s}_{i+1}$. It is not difficult to verify that these choices define a trajectory τ in Q that corresponds to the assumed trajectory $\bar{\tau}$ in P. □

The full correspondence between infinite π-trajectories τ in Q and infinite non-terminating π-trajectories τ in $P = T_D(Q)$ suggests the following definition of *termination* in QNPs Q and FOND problems $T_D(Q)$:

Definition 12 (Termination in Q)**.** *A policy π for the QNP Q is terminating iff all the π-trajectories on Q are of finite length. In such a case, we say that π is Q-terminating.*

Definition 13 (Termination in P)**.** *A policy π for the FOND problem $P = T_D(Q)$ is terminating iff all the infinite π-trajectories on P are terminating. In such a case, we say that π is P-terminating.*

The correspondence between policies can then be expressed as:

Theorem 14. *Let Q be a QNP, let $P = T_D(Q)$ be its direct translation, and let π be a policy for Q (and thus also for P). Then, π is Q-terminating iff π is P-terminating.*

Proof. Direct from Theorem 11. For one direction, assume that π is Q-terminating and let $\bar{\tau}$ be a π-trajectory in P. If $\bar{\tau}$ is not terminating, by Theorem 11, τ is infinite and thus π would not be Q-terminating. Therefore, every π-trajectory $\bar{\tau}$ in P is terminating and thus π is P-terminating. The other direction is established similarly. □

The *soundness and completeness* of the direct translation extended with termination can be expressed as following:

Theorem 15 (Soundness and Completeness T_D)**.** *Let Q be a QNP, let $P = T_D(Q)$ be its direct translation, and let π be a policy for Q (and thus also for P). The following are equivalent:*

1. π solves Q,

2. π is a strong cyclic solution of Q and π is Q-terminating,

3. π is a strong cyclic solution of P and π is P-terminating.

Proof. (1 $\Leftrightarrow$ 2) Assume that π solves Q. If there is π-trajectory connecting an initial state with state s, there must be π-trajectory connecting s with a goal state. Otherwise, π would not be a solution for Q. Likewise, if τ is an infinite π-trajectory in Q, then τ does not reach a goal state and thus π would not solve Q. For the converse direction, assume that π is a strong cyclic solution for Q and that π is Q-terminating, and suppose that π does not solve Q. Then, there is a maximal π-trajectory τ in Q that does not reach a goal state. It cannot be the case that τ ends in a state s where $\pi(s)$ is undefined or non-applicable as π then would not be a strong cyclic solution for Q. Hence, τ must be infinite but this contradicts the assumption that π is Q-terminating.

(2 $\Leftrightarrow$ 3) By Theorem 8, π is a strong cyclic solution for Q iff it is strong cyclic solution for P. By Theorem 14, π is Q-terminating iff it is P-terminating. □

5. Checking Termination with Sieve

Sieve is the procedure introduced by Srivastava et al. (2011) to test whether a policy terminates. It runs in time that is polynomial in the number of states of the FOND problem reached by the policy.[3] For this, the algorithm takes as input a policy graph $\mathcal{G}(P,\pi) = \langle V, E\rangle$ constructed from the FOND problem $P = T_D(Q)$ and a strong cyclic policy π for P. The nodes in the policy graph are the states $\bar{s}$ in the state model $\mathcal{S}(P)$ that are reachable from the initial state and the policy π, and the directed edges in E are the pairs $(\bar{s}, \bar{s}')$ for $\bar{s}' \in F(a, \bar{s})$ and $\pi(\bar{s}) = a$. These edges are labeled with the action a. The algorithm iteratively removes edges from the graph $\mathcal{G}(P,\pi)$ until the graph becomes acyclic or no additional edge can be removed.

For incrementally removing edges from the graph, Sieve identifies first its *strongly connected components* by a single depth-first search traversal, following Tarjan's algorithm (Tarjan, 1972). A strongly connected component (SCC) is a partition of the nodes of the graph such that if a node $\bar{s}$ belongs to a partition, any node $\bar{s}'$ that can be reached from $\bar{s}$ and that can reach $\bar{s}$ back in the graph, is placed in the same partition as $\bar{s}$.

The algorithm then picks a variable X and a SCC such that the variable X is decremented but not incremented in the SCC. That is, there must be a state $\bar{s}$ in the SCC such that $\pi(\bar{s})$ is a $Dec(X)$ action and no $\pi(\bar{s}')$ is an $Inc(X)$ action for any $\bar{s}'$ in the SCC. The algorithm then removes all the edges $(\bar{s}, \bar{s}')$ in the SCC such that $\pi(\bar{s})$ is a $Dec(X)$ action. We abbreviate this by saying that *variable X is removed from the SCC*, which means that the edges associated with $Dec(X)$ actions are removed. Following the edge removals, the SCCs must be recomputed and the process is repeated until the graph becomes acyclic or no more edges can be removed in this manner. The result is that:

Theorem 16. *Let Q be a QNP. A policy π for the FOND problem $P = T_D(Q)$ is P-terminating iff* Sieve *reduces the policy graph $\mathcal{G}(P,\pi)$ to an acyclic graph.*

Proof. We show the contrapositive of the two implications that make up the equivalence. First, let us assume that Sieve terminates with a cyclic graph $\mathcal{G}$ and, thus, let C be a non-trivial SCC in the graph $\mathcal{G}$. Since every state in $\mathcal{G}$ is reachable from the initial state by π and since every state in C is also reachable by π from any other state in C, there is a π-trajectory $\bar{\tau}$ in P of the form $\bar{s}_0, \ldots, [\bar{s}_i, \ldots, \bar{s}_m]^*$ where the recurrent states $\bar{s}_i, \ldots, \bar{s}_m$ in $\bar{\tau}$ are *exactly* all the states in C. Observe that Sieve is unable to remove any further edge from $\mathcal{G}$. If no variable is decremented in the loop, $\bar{\tau}$ is not P-terminating by definition. Similarly, if some variable X is decremented in the loop (i.e., C), X is also incremented in the loop and the trajectory is again not P-terminating.

3. Our account of termination differs from the one of Srivastava et al. (2011) in two main aspects. First, our notion of termination is articulated independently of the algorithm for checking termination. Second, our version of the algorithm is developed and applied to QNPs that involve both numerical and boolean variables.

Sieve (Graph $\mathcal{G} = \mathcal{G}(P, \pi)$):
repeat
  Compute the strongly connected components (SCC) of $\mathcal{G}$.
  Choose an SCC C and a variable X that is decreased in C but is not increased in C;
  i.e., for some $\bar{s}$ in C, $\pi(\bar{s})$ is a $Dec(X)$ action, and for no $\bar{s}$ in C, $\pi(\bar{s})$ is an $Inc(X)$ action.
  Remove the edges $(\bar{s}, \bar{s}')$ such that $\bar{s}$ and $\bar{s}'$ are in C, and $\pi(\bar{s})$ is a $Dec(X)$ action.
until $\mathcal{G}$ *is acyclic (terminating) or there is no SCC* C *and variable* X *to choose (non-terminating)*

Figure 1: Sieve procedure for testing whether policy π for FOND problem $P = T_D(Q)$ terminates (Srivastava et al., 2011).

For the other implication, let us assume that π is *not* P-terminating. Then, there must be an infinite π-trajectory $\bar{\tau}$ in P of the form $\bar{s}_0, \ldots, [\bar{s}_i, \ldots, \bar{s}_m]^*$ where every variable that is decremented by an action $\pi(\bar{s}_j)$, $j \in [i, m]$, is incremented by another action $\pi(\bar{s}_j)$, $j \in [i, m]$. We want to show that Sieve terminates with a graph that has one SCC that includes all the states in the loop. Indeed, initially, all the states in the loop must be in one component C as they are all reachable from each other. Observe that edges that correspond to actions that do not decrement variables are not removed by Sieve. Hence, if the loop does not decrement any variable X, the states in the loop cannot be separated into different SCCs. On the other hand, Sieve cannot remove a variable X from the loop as the component C features actions that increment and decrement X. Therefore, as before, the states in the loop stay together within an SCC for the whole execution of Sieve. □

The Sieve procedure, slightly reformulated from Srivastava et al. (2011), is depicted in Figure 1.

Example. The policy π for Q_{nest} above is given by the rules:

$$\text{if } X > 0 \text{ and } Y = 0, \text{ then do } a\,, \tag{13}$$

$$\text{if } X > 0 \text{ and } Y > 0, \text{ then do } b \tag{14}$$

where recall that $Q_{nest} = \langle F, V, I, O, G\rangle$ with $F = \emptyset$, $V = \{X, Y\}$, $I = \{X > 0, Y > 0\}$, $G = \{X = 0\}$, and $O = \{a, b\}$ where $a = \langle X > 0, Y = 0; X\downarrow, Y\uparrow\rangle$ and $b = \langle Y > 0; Y\downarrow\rangle$. The policy decrements Y using the action b until the action a that decreases X and increases Y can be applied. The process is repeated until $X = 0$. The nested loops in the policy graph $\mathcal{G}(P, \pi)$ are shown in Figure 2. The policy graph contains three states: the leftmost one is the initial state, and the rightmost one is a goal state. The two states on the left are reachable from each other, and hence, define a strongly connected component (SCC). In this SCC, the $Dec(X)$ edges are removed by Sieve because X is not increased anywhere. Once this is done, the $Dec(Y)$ edges are removed by Sieve because the edges associated with $Inc(Y)$ effects are gone. The resulting graph is acyclic establishing thus that the policy π terminates in $P = T_D(Q_{nest})$. □

Using Sieve it is easy to see that the problem of checking the existence of plans for QNPs can be decided in exponential space:

Theorem 17 (Srivastava et al., 2011)**.** *Deciding plan existence for QNPs is in* EXPSPACE.

Proof. Let $Q = \langle F, V, I, O, G\rangle$ be a QNP. The number of boolean states for Q is exponential in the number of fluents and variables; i.e., $|F| + |V|$. A policy π for Q can be described in exponential

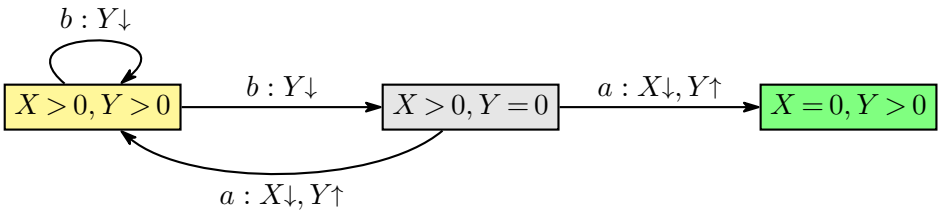


Figure 2: Testing termination with Sieve. Policy graph $\mathcal{G}(P, \pi)$ for the FOND problem $P = T_D(Q_{nest})$ and policy π, from the example in text, containing three states: the leftmost is the initial state, and the rightmost is a goal state. The two states on the left are reachable from each other, and hence, define a strongly connected component (SCC). In this SCC, the $Dec(X)$ edges are removed by Sieve because X is not increased anywhere. Once this is done, the $Dec(Y)$ edges are removed because the edges associated with $Inc(Y)$ effects have been eliminated. The resulting graph is acyclic and hence π terminates in P.

space as a mapping from boolean states into actions. A brute-force algorithm enumerates all policies one by one using exponential space. Each policy is tested for strong cyclicity and termination. The former is a straightforward test in a graph while the latter is done with Sieve. If the policy is strong cyclic and terminating, Q is accepted. Otherwise, if no policy is found to be strong cyclic and terminating, Q is rejected. Since testing strong cyclicity and running Sieve both require polynomial time in the size of the input policy, the whole algorithm can be implemented in space that is exponential in the size of Q. □

Below we improve this bound and show exponential time (EXP) solvability through a more complex translation of QNPs into FOND problems that is also polynomial. The novelty of the new translation is that the strong-cyclic policies of the resulting FOND problems do not need to be checked for termination. QNPs are thus *fully reduced* to FOND problems. Since FOND problems can be reduced to QNPs as well, *we will show indeed that FOND problems and QNPs have the same expressive power*, and the complexity of plan existence for FOND problems is known to be EXP-Complete (Littman et al., 1998; Rintanen, 2004), then these reductions show the EXP-Completeness of the plan existence decision problem for QNPs. In addition to the establishment of novel theoretical results, these reductions are also of practical importance as they permit the computation of solutions; once a QNP is reduced to FOND, a solution for the QNP can be recovered in linear time from a solution to the FOND problem, and the same for the reduction from FOND problems into QNPs.

The distinction between the classes EXP and EXPSPACE is important. EXPSPACE contains the (decision) problems that can be solved with Turing machines (TMs) that operate in exponential space (as a function of the input size), yet such TMs may in fact run in doubly exponential time as the running time is bounded, in the worst case, by an exponential function of the bound in space (Sipser, 2006). On the other hand, EXP comprises the problems that can be solved with TMs that run in exponential time. The difference is analogous to the difference between the classes P (polynomial time) and PSPACE (polynomial space).

6. First Reduction: FOND problems into QNPs

The first reduction that we introduce is from FOND problems into QNPs. It is a non-trivial reduction yet simpler than the inverse reduction from QNPs into FOND problems. The main

obstacle to overcome is that the non-deterministic effects in FOND problems are over boolean variables, while those in QNPs are only on numerical variables through decrements. Another important obstacle is that strong cyclic solutions in QNPs are not QNP solutions unless they are terminating.

Let $P = \langle F, I, O, G\rangle$ be a FOND problem and let us denote the non-deterministic effects of action a as $E^a_1 \,|\, E^a_2 \,|\, \cdots \,|\, E^a_{k_a}$ where each E^a_i is a set (conjunction) of F-literals, and k_a denotes the number of non-deterministic effects of the action a. If $k_a = 1$, the action a is deterministic, else it is non-deterministic. For simplicity, we assume that the set of effects $\{E^a_i\}_i$ for action a aggregates all the multiple non-deterministic effects in the description of a in P, and the reduction below is presented under this assumption. Afterwards, we discuss how to handle in polynomial time FOND problems whose transitions are factorized.

We map P into a QNP $Q = \langle F', V', I', O', G'\rangle$ that extends P with numerical variables $V' = \{X\} \cup \{Y_{a,i} : a \in O, 1 \leq i \leq k_a\}$, extra boolean variables, and extra actions; i.e., $F \subseteq F'$, $I \subseteq I'$, $O \subseteq O'$, and $G' = G$.

The heart of the reduction lies in the way in which the non-deterministic effects of each action are captured in Q. For this, the collection of non-deterministic effects of the action a are replaced by an $Inc(X)$ action, for the unique numerical variable X, followed by a *fixed loop* where the variable X is decremented until becoming zero. The alternative effects E^a_i are then triggered when $X = 0$ becomes true in the corresponding part of the loop. Intuitively, each non-deterministic action a becomes a "wheel of fortune" that must be spun to select the non-deterministic effect to be applied. The increments and decrements of the extra variables $Y_{a,i}$ ensure that the strong cyclic policies π for P, and only those, induce policies π' for Q that are terminating. The fixed loop sequence associated with action a performs the following steps, that are implemented with the help of new auxiliary actions and propositional symbols:

1. $Inc(X)$ (implemented by modified action a),
2. $Dec(X)$ (implemented by new action $Spin$),
3. If $X = 0$, apply the effects in E^a_1, increment $Y_{a,1}$, decrement $Y_{a,j}$, $j \neq 1$, and break loop (implemented by new actions $Prep(a, i)$ and $Exit(a, i)$ for $i = 1$),
4. $Dec(X)$ (implemented by new action $Next(a, i)$ for $i = 1$)
5. If $X = 0$, apply the effects in E^a_2, increment $Y_{a,2}$, decrement $Y_{a,j}$, $j \neq 2$, and break loop (implemented by new actions $Prep(a, i)$ and $Exit(a, i)$ for $i = 2$),
6. $Dec(X)$ (implemented by the new action $Next(a, i)$ for $i = 2$)

$\vdots$

7. If $X = 0$, apply the effects in $E^a_{k_a}$, increment Y_{a,k_a}, decrement $Y_{a,j}$, $j \neq k_a$, and break loop (implemented by new actions $Prep(a, i)$ and $Exit(a, i)$ for $i = k_a$),
8. $Dec(X)$ and go back to 3 (implemented by new action $Loop(a)$)

If, throughout the loop, some variable $Y_{a,j}$ becomes zero, the next action to apply is forced to be a new action that achieves the goal thus ending the execution. This mechanism takes care of the non-fair trajectories that may exist in P.

To show that the resulting mapping is indeed a reduction (i.e., the mapping is sound and complete), two things need to be established: that a policy π that solves Q induces a policy π' that

solves P (soundness), and vice versa, that a policy π' that solves the FOND problem P induces a policy π that solves Q (completeness).

The fixed sequence of effects for each action a in Q is implemented using the following additional boolean variables:

- A boolean *normal* that is false when the sequence is entered for some action a and made true when the loop is exited.
- A boolean $ex(a)$ to express that the sequence for the action a is being executed. The reduction is such that the atoms in $\{normal\} \cup \{ex(a) : a \in O\}$ are pairwise mutex and one of them is always true.
- A counter from 0 to K encoded using mutex atoms $cnt(\ell)$, $\ell = 0, 1, \ldots, K+1$, that is set to 1 when the loop is entered and re-entered (step 8 above) and where K is the maximum number of non-deterministic outcomes of any action in P. (An engineering trick that saves an extra boolean permits $cnt(0)$ and $cnt(i)$ to be both true after executing the action $Prep(a, i)$ below.)

The actions that implement the fixed loop sequence (i.e., spin the wheel) are the following:

- $Spin = \langle \neg normal, cnt(0), X > 0; \neg cnt(0), cnt(1), X\downarrow \rangle$.
- $Next(a, i) = \langle ex(a), cnt(i), X > 0; \neg cnt(i), cnt(1+i), X\downarrow \rangle$ to advance along the fixed loop sequence while $X > 0$ and decrementing X.
- $Loop(a) = \langle ex(a), cnt(k_a), X > 0; \neg cnt(k_a), cnt(1), X\downarrow \rangle$ to start a new iteration of the loop.

These are the only actions that decrease the variable X, while the (modified) non-deterministic actions being the only actions that increase X (see below). Once X becomes zero, the non-deterministic effect to apply is determined by the value of the counter. The actions that apply the selected effect and capture the non-fair trajectories are:

- $Prep(a, i) = \langle ex(a), cnt(i), \neg cnt(0), X = 0, Y_{a,j} > 0; cnt(0), Y_{a,i}\uparrow, \{Y_{a,j}\downarrow\}_{j \neq i} \rangle$ where the precondition $Y_{a,j} > 0$ is for all j, and the effect $Y_{a,j}\downarrow$ for all $j \neq i$.
- $Exit(a, i) = \langle ex(a), cnt(i), cnt(0), X = 0, Y_{a,j} > 0; \neg ex(a), \neg cnt(i), normal, E^a_i \rangle$ where the precondition $Y_{a,j} > 0$ is for all j.
- $Fin(a, j) = \langle ex(a), X = 0, Y_{a,j} = 0; \neg ex(a), \{\neg cnt(i)\}_{i \geq 1}, normal, G \rangle$ that is forced when $Y_{a,i} = 0$ and reaches the goal G, where the effect $\neg cnt(i)$ is for all positive i.

The only actions that affect the variables $Y_{a,j}$ are the $Prep(a, i)$ actions. After applying such an action, some (one or more) variables $Y_{a,j}$ may become zero. If so, $Fin(a, j)$ becomes the only applicable action and the execution terminates as the goal is reached. If no such variable becomes zero, $Exit(a, i)$ becomes forced which applies the effect E^a_i and restores "normal" operation.

The initial state of the QNP includes the atoms *normal* and $cnt(0)$. Deterministic actions in P "pass directly" into the QNP Q with these two atoms added as extra preconditions. Non-deterministic actions a, however, are handled differently by replacing their effects $E^a_1 \mid E^a_2 \mid \cdots \mid E^a_{k_a}$ by deterministic effects $\{\neg normal, ex(a), X\uparrow\}$ after which the only applicable action would be the *Spin* action from above. The idea of the construction is illustrated in Figure 3.

Definition 18. *Let $P = \langle F, I, O, G \rangle$ be a FOND problem. The reduction R maps P into the QNP $Q = \langle F', V', I', O', G' \rangle$ given by*

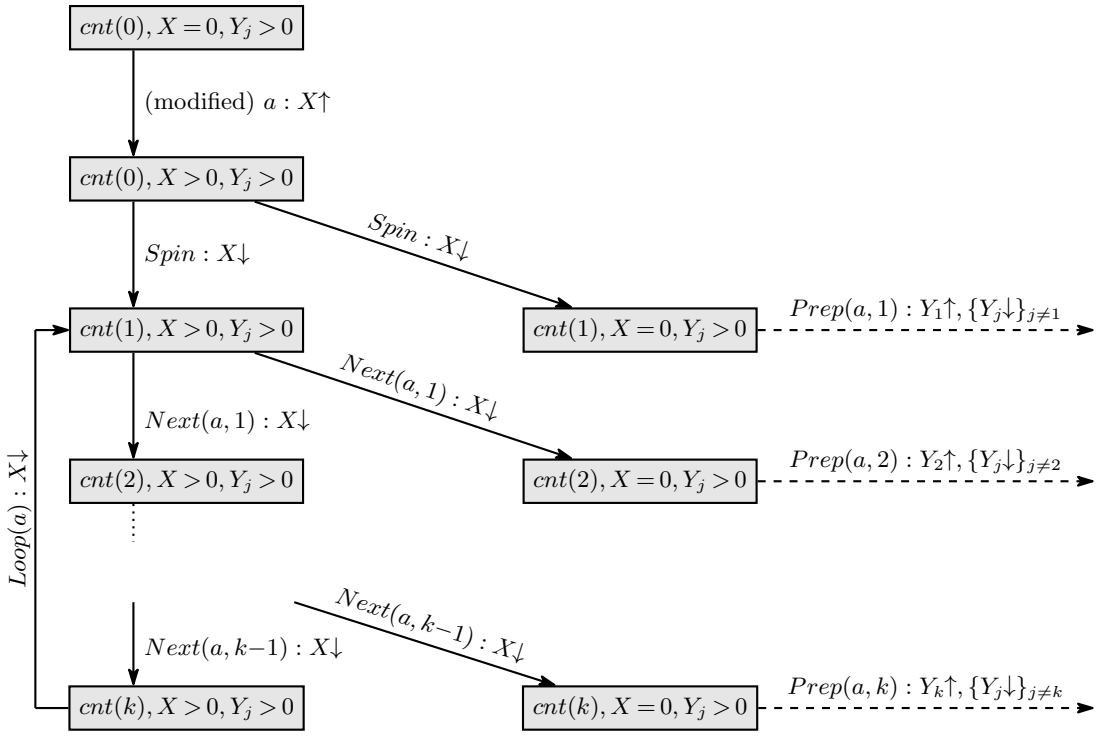


Figure 3: Encoding the non-deterministic boolean effects $E_1 \mid \cdots \mid E_k$ of an action a in the FOND problem P as a fixed sequence of effects that loops while decrementing the variable X in the QNP $Q = R(P)$. The variable X implements the loop while the counter $cnt(i)$ determines which effect E_i obtains when X becomes zero. When $X = 0$ and $cnt(i)$ hold, $Prep(a, i)$ is applied whose function is to increment $Y_i = Y_{a,i}$ and decrement the other Y_j variables; if no such variable becomes zero, the effect E_i is applied with the action $Exit(a, i)$ (not shown), otherwise $Fin(a, j)$ is applied when Y_j becomes zero (not shown). The variables Y_i are used to map *unfair* trajectories in P into goal-reaching trajectories in Q, while forcing solutions of Q to induce solutions for P. Indeed, if one unfair trajectory contains the action a infinitely often but neglects (starves) the effect E_i, the variable Y_i eventually becomes zero, since the only action that increments it is $Prep(a, i)$, and then the trajectory forcibly applies the action $Fin(a, i)$ that terminates the execution by reaching the goal (see text for details).

1. $F' = F \cup \{normal\} \cup \{ex(a) : a \in O\} \cup \{cnt(\ell) : \ell \in [0, K]\}$ *where* $K = \max_{a \in O} k_a$,
2. $V' = \{X\} \cup \{Y_{a,i} : a \in O, i \in [1, k_a]\}$,
3. $I' = I \cup \{normal, cnt(0)\} \cup \{X = 0\} \cup \{Y_{a,i} > 0 : a \in O, i \in [1, k_a]\}$,
4. $O' = O \cup \{Spin\} \cup \{\mathcal{A}(a, i) : \mathcal{A} \in \{Prep, Exit, Fin, Next\}, a \in O, i \in [1, k_a]\} \cup \{Loop(a) : a \in O\}$,
5. $G' = G$.

For stating the formal properties of the reduction, we need to specify how a policy π for Q induces a policy π' for P, and vice versa, as Q involves extra variables and actions.

Let us split the *non-goal states* in Q into two sets: the normal states, where the booleans *normal* and $cnt(0)$ are true, and the rest of non-goal states where *normal* or $cnt(0)$ is false. (Observe that there cannot be a normal state where some variable $Y_{a,j}$ is zero as when that happens, the loop must exit with $Fin(a, j)$ that reaches a goal state.) In the normal states, the (qualitative) value of all the extra variables is the same: all the extra boolean variables are false except for *normal* and $cnt(0)$ that are true, $X = 0$, and $Y_{a,i} > 0$ for $a \in O$ and $i = 1, 2, \ldots, k_a$. Hence, any policy for P can be converted into a policy for Q, and vice versa. For example, a policy π for P translates directly into a policy for Q on normal states, where the actions for non-normal states is determined (the only real choice is when more that one variable $Y_{a,j}$ is zero, but then any choice of the $Fin(a, j)$ actions is equivalent as such actions yield goal states). Conversely, any policy for Q induces a unique policy for P as each non-goal state in P is associated with normal states in Q that get the same action from the policy. The main properties of the reduction are expressed as follows:

Theorem 19 (Reduction FOND to QNP)**.** *The mapping R is a polynomial-time reduction from FOND problems into QNPs. That is, a FOND problem P has a solution iff the QNP $Q = R(P)$ has a solution. Moreover, a solution for P (resp. Q) can be recovered in polynomial time from a solution for Q (resp. P).*

Proof. It is straightforward to see that R is computable in time that is polynomial in the size of the input P and the maximum value k_a for any action a. (See below for a discussion on how to deal with FOND problems that may have multiple non-deterministic effects per action.) Likewise, as explained above, policies for P map into policies for Q, and vice versa.

We first show how to map infinite fair trajectories in P into trajectories in Q that visits normal states infinitely often, and vice versa. Indeed, for an infinite trajectory τ in P, we can construct a corresponding trajectory τ' in Q by augmenting the states in τ to give value to the extra booleans and numerical variables in Q, and by "inserting" sub-trajectories over non-normal states for the triplets (s, a, s') in τ for non-deterministic actions a. Indeed, for such a triplet, it is enough to determine the effect E_i^a that produces the successor state s' in order to obtain a finite sequence of non-normal states that map s into s' in Q. For the converse, if τ' is a trajectory in Q that visits normal states infinitely often, then τ' does not contain an action $Fin(a, i)$. Thus, we can construct a trajectory over P by just projecting τ' over its normal states. This trajectory is fair since if some effect E_i^a is starved from an action a that is applied infinitely often, the variable $Y_{a,i}$ would become zero forcing the application of $Fin(a, i)$.

For showing the first implication, let us consider a policy π for P. We want to show that the policy π' obtained from π solves Q. Indeed, suppose that π' does not solve Q. Then, π' is either not strong cyclic or non terminating in Q. The former is impossible since π connects each reachable normal state to a goal state. Thus, π' is non terminating in Q and there is a non-terminating trajectory $\bar{s}_0, \ldots [\bar{s}_i, \ldots, \bar{s}_m]^*$ in $T_D(Q)$. The loop increments and decrements the variable X, and all the variables $Y_{a,j}$ for the actions a that are applied in the loop, and thus it must contain some normal state. Therefore, we can construct an infinite fair π-trajectory in P that is non-goal reaching contradicting the assumption that π solves P.

For showing the second implication, let π' be a policy for Q and let us suppose that the policy π induced from π' does not solve P. Then, either π' is not defined over its reachable states or there is an infinite fair π-trajectory in P, one that does not reach the goal. The first case is impossible since π' solves Q. For the second case, if such trajectory exists, then there is a π'-trajectory τ in Q that visits normal states infinitely often. This trajectory is associated with a non-terminating trajectory in $T_D(Q)$ in contradiction with π' being a solution for Q. □

Theorem 19 states that the strong cyclic policies for a FOND problem P correspond exactly with the policies of the QNP $Q = R(P)$ that can be obtained from P in polynomial time. There is an analogous translation for computing the *strong policies* of P; namely, the strong cyclic policies that are actually acyclic (no state visited twice along any trajectory). For this, let R' be the translation that is like R above but with the numerical variables $Y_{a,i}$ removed; i.e., initial and goal conditions on $Y_{a,i}$ are removed from $R'(P)$ as well as the conditions and effects on $Y_{a,i}$ in the actions $Prep(a, i)$ and $Exit(a, i)$, and with the actions $Fin(a, j)$ removed.

Theorem 20 (Reduction Strong FOND to QNP)**.** *The mapping R' is a polynomial time reduction from FOND problems with strong solutions into QNPs. That is, a FOND problem P has a strong solution iff the QNP $Q = R'(P)$ has solution. Moreover, a strong solution for P (resp. solution for Q) can be recovered in polynomial time from a solution for Q (resp. strong solution for P).*

Proof. In the absence of the variables $Y_{a,i}$, the only solutions to $Q = R'(P)$ must be acyclic, as any cycle would involve increments and decrements of the single numerical variable X, and thus, would not be terminating. The rest of the proof follows from the correspondence laid out in the previous proof between the trajectories over the normal states in Q and the trajectories in P. □

Let us now consider the case when the FOND problem P contains actions with multiple non-deterministic effects. Let a be one such action, and let n be the number of non-deterministic effects in a, each one denoted by $E_1^j \mid \cdots \mid E_{k_j}^j$ where $j \in [1, n]$ and k_j is the number of outcomes for the j-th non-deterministic effect of a. By assumption, every choice of outcomes for the effects yields a *consistent* (aggregated) outcome for a (cf. footnote 1). The easiest way to accommodate such actions is to "preprocess" P by replacing the action a by the *sequence* $\langle a(1), a(2), \ldots, a(n)\rangle$ of n non-deterministic actions, each one featuring exactly one effect; i.e., the effect of $a(j)$ is $E_1^j \mid \cdots \mid E_{k_j}^j$. For this, the precondition of $a(1)$ is set to the precondition of a, that of $a(j)$ to $\{seq(a), next(j)\}$, $j \in [2, n]$, and the precondition of every other action is extended with $\neg seq(a)$. Likewise, the effect of $a(1)$ is extended with $\{seq(a), next(2)\}$, that of $a(j)$ with $\{\neg next(j), next(j+1)\}$, $j \in [2, n-1]$, and that of $a(n)$ with $\{\neg next(n), \neg seq(a)\}$. The new atoms $seq(a)$ and $next(j)$ denote that the sequence for a is being applied and the next action to apply in the sequence is $a(j)$ respectively. In this way, the FOND problem P is converted in linear time into an *equivalent* FOND problem where each action has exactly one non-deterministic effect. In other words, we may assume without loss of generality that the FOND problem P is such that each action has at most one non-deterministic effect since if this is not the case, P can be converted into an equivalent FOND problem P' in linear time. By equivalent, we mean that any solution π for P can be converted in polynomial time into a solution π' for P', and vice versa.

We have shown how strong cyclic and strong planning over a FOND problem P translates into QNPs: in one case, including the extra variables X and $\{Y_{a,i}\}_{a,i}$ in place, in the second, only X. The translation with these variables offers the possibility of capturing more subtle forms of fairness. For example, if we just remove from the translation a variable $Y_{a,i}$ along with the effects on it, the resulting QNP would assume that all the outcomes E_j, $j \neq i$, of action a are fair (i.e., they cannot be skipped forever in a fair trajectory) but that the outcome E_i is not. In other words, while in strong cyclic planning, all the non-deterministic actions are assumed to be fair, and in strong planning, all of them to be unfair, in QNP planning, it is possible to handle a combination of fair and unfair actions (as in dual FOND planning, Geffner & Geffner, 2018), as well as a combination of fair and unfair outcomes of the same action.

7. Second Reduction: QNPs into FOND problems

We have shown that FOND problems can be reduced in polynomial time to QNPs. We show now the other direction: QNPs can be reduced in polynomial time to FOND problems. The two results imply a new complexity result; namely, that QNPs have the same expressive power as FOND problem and that the plan existence decision problem for both models have the same complexity. This second translation T is more subtle than the first and unlike the direct translation T_D above, it is a full reduction which does not require termination tests.

The first attempt at such a translation was sketched by Bonet et al. (2017) but the reduction is buggy as it is not always sound. The intuition, however, is useful and we build on it. Basically, that reduction introduces boolean variables q_X that when set to true preclude increments of variable X, hence making the decrements of X "fair". The variable q_X can be reset to false when the loop "finishes", i.e., when $X = 0$ is true. This idea, however, does not fully avoid non-terminating loops and hence, by itself, does not produce a sound reduction.[4] The *new translation* replaces the q_X variables by a *bounded stack* that keeps the variables X that are being decremented *in order*, and suitable *counters*. The new variables and actions enforce that solutions of the FOND problem $P = T(Q)$, unlike the solutions of the direct translation $T_D(Q)$, are all terminating. For this, the new translation introduces conditions that mirror those captured by the Sieve procedure. In particular, for capturing policies that terminate, the variables are to be placed on the stack following the order by which Sieve removes them.

7.1 Extra Variables and Actions

The reduction $T(Q)$ introduces a bounded stack α where numerical variables from V can be pushed and popped, and bounded counters $c(d)$, for $d = 0, \ldots, |V|$ that are associated with the possible levels (depths) $d = |\alpha|$ of the stack. There is also a top counter c_T that may only increase. The stack starts empty and may grow to contain all the variables in V, but no variable can appear in the stack more than once. The stack is represented as growing from left to right; e.g., αX is the stack that results of pushing the variable X in the stack α. The c counters start at 0 and may grow up to a Max number, that for completeness must be set to $1 + 2^n$ where n is the total number of boolean and numerical variables in Q. In practice, Max can be set to a much small number.[5] In any case, the counters and the stack are captured in terms of a polynomial number of boolean variables and the whole reduction $T(Q)$ is computed in polynomial time. The state of the stack α is captured by the atoms $in(X)$, $depth(d)$, and $index(X, d)$ that represent whether X is in the stack, the depth of the stack, and the depth at which X is in the stack, respectively. X is the top element in the stack when $index(X, d)$ and $depth(d)$ are true (i.e., the stack is αX and $|\alpha| = d - 1$),

4. Consider a QNP $Q = \langle F, V, I, O, G\rangle$ with a single numerical variable X and four actions a, b, c, and d that result in a loop where $a = \langle p_1, X > 0; \neg p_1, p_2, X\downarrow\rangle$, $b = \langle p_2; p_3, \neg p_2\rangle$, $c = \langle p_3, X > 0; X\downarrow\rangle$, $d = \langle p_3, X = 0; \neg p_3, p_1, X\uparrow\rangle$. Let us assume that $I = \{p_1, X > 0\}$ and $G = \{X = 0, p_2\}$. There is a single policy π for Q, as in all the (non-goal) states that can be reached from I, there is a single applicable action. This policy π is strongly cyclic but is not terminating. The reason is that one of the trajectories determined by the policy is a non-terminating loop $s_0, a, s_1, b, s_2, c, s_3, d, s_0, \ldots$ where the single variable that is decremented (X) is also incremented, and where $\bar{s}_0 = \{p_1, X > 0\}$, $\bar{s}_1 = \{p_2, X > 0\}$, $\bar{s}_2 = \{p_3, X > 0\}$, and $\bar{s}_3 = \{p_3, X = 0\}$. The FOND problem that results from the translation sketched by Bonet et al. (2017) accepts this policy π as a solution, which is incorrect. This happens because the variable q_X can be set and reset an infinite number of times; indeed, right before and right after the action c in the loop, respectively. The new translation excludes such non-terminating loops via a stack and counters.

5. For structured policies that result in loops that can be entered and left through single entry and exit points, Max is the bound on the number of consecutive loops (blocks), possibly with other loops nested, that the policy can generate at the same level.

and it is bottom element when $index(X, 1)$ is true (i.e., the stack is X). The stack is empty when $depth(0)$ holds.

The extra actions in $P = T(Q)$ are those for pushing and popping variables to and from the stack, and for advancing the top counter c_T.

1. **Actions** $Push(X, d)$ for variable X and depth $d \in [0, |V| - 1]$ have preconditions $\neg in(X)$, $depth(d)$ and $c(d) < Max$, and effects:

   a) $in(X)$, $index(X, d + 1)$, $depth(d + 1)$ and $\neg depth(d)$ to push X and increase stack depth,

   b) $c(d) := c(d) + 1$ to increment counter for old level,

   c) $c(d + 1) := 0$ to initialize counter for new level.

2. **Actions** $Pop(X, d)$ for variable X and depth $d \in [1, |V|]$ have preconditions $in(X)$, $index(X, d)$ and $depth(d)$, and effects:

   a) $\neg in(X)$, $\neg index(X, d)$, $\neg depth(d)$, $depth(d - 1)$ to pop X and decrease stack depth.

3. **Action** $Move$ advances the top counter c_T by 1 when the stack is empty; i.e., it has preconditions $depth(0)$ and $c_T < Max$, and effect $c_T := c_T + 1$.

For simplicity, we assume that the language of our FOND problems P makes room for the integer counters $c(d)$, $d = 0, \ldots, |V|$, that may be increased by 1, from 0 up to a fixed number Max and that may be reset back to 0. In the implementation, these counters are represented in terms of a linear number of boolean variables.[6]

The actions a that belong to Q are split into two classes. Actions *a that do not decrement any variable*, keep their names in $P = T(Q)$ but replace their $Inc(X)$ effects by propositional effects $X > 0$, and add the precondition $\neg in(X)$ that disables a when X is in the stack:

4. **Actions** a **in** Q **that decrement no variable**, keep the same names in $T(Q)$, the same preconditions and same effects, except that the effects $Inc(X)$ are replaced by propositional effects $X > 0$, if any, and in such a case, the precondition $\neg in(X)$ is added.

Actions a from Q *that decrement variables* (and hence introduce non-determinism) map into actions in P of type $a(X, d)$ where X is a variable decremented by X that is in the stack at depth d: more of the variables that are decremented are in the stack when the action is applied:

5. **Actions** $a(X, d)$ **for** a **in** Q **that decrement a variable** X **in the stack at level** d **(and possibly others)** inherit propositional preconditions and effects of a, and for each variable Y that is increased by a, they include the precondition $\neg in(Y)$ and the effect $Y > 0$. The parameter $d \in [1, |V|]$ stands for the current stack depth. The action $a(X, d)$ also has:

   a) extra precondition $index(X, d)$ (i.e., d is level at which X appears in stack),

6. In the actual encoding, the counters $c(d)$ are represented with $1 + n$ atoms (bits), $b_i(d)$, $i = 0, \ldots, n$, where n is total number of variables in Q, i.e., $n = |F| + |V|$. A preconditions such as $c(d) < Max$ then translates into the precondition $\neg b_n(d)$; the least and most significant bits for $c(d)$ are $b_0(d)$ and $b_n(d)$ respectively. Increments of $c(d)$ by 1 may be translated in two different ways, either by using conditional effects, or by increasing the number of actions. For the former, conditional effects of the form $b_0(d), \ldots b_{i-1}(d), \neg b_i(d) \rightarrow \neg b_0(d), \ldots, \neg b_{i-1}(d), b_i(d)$, for $i \in [0, n]$, are used. For the latter, each action act that increases $c(d)$ is replaced by n actions $act(i)$, $i \in [0, n]$, that are like act but have the extra precondition $b_0(d), \ldots b_{i-1}(d), \neg b_i(d)$ and the extra effects $b_0(d), \ldots, \neg b_{i-1}(d), b_i(d)$. The first translation, however, when compiled into STRIPS introduces additional actions as well (Nebel, 2000). Finally, a reset effect $c(d) := 0$ is obtained by setting all atoms $b_0(d)$, $i \in [0, n]$, to false.

b) extra non-deterministic effects $X_i > 0 \,|\, X_i = 0$ for each $Dec(X_i)$ effect, and

c) extra effects $c(d') := 0$ for each d' such that $d \leq d' \leq |V|$ to reset the counters for the levels above or equal to d.

In words, actions a from Q that do not decrement any variable map into a single action of the form a in $P = T(Q)$, while actions a from Q that decrement variables map into actions $a(X, d)$ applicable only when a variable X decremented by a is in the stack at level d. The actions of the form a in $P = T(Q)$ are deterministic, and only actions $a(X, d)$ can generate cycles in a strong cyclic policy for P. The reduction $P = T(Q)$ can be summarized as follows:

Definition 21 (Reduction QNP to FOND)**.** *Let $Q\langle F, V, I, O, G\rangle$ be a QNP. The FOND problem $P = T(Q)$ is $P = \langle F', I', O', G'\rangle$ with:*

1. $F' = F \cup \{c_T\} \cup \{depth(d), c(d)\} \cup \{in(X)\} \cup \{index(X, d')\}$,
2. $I' = I \cup \{depth(0), c_T = 0, c(0) = 0\}$,
3. $G' = G$,
4. $O' = \{a : a \in O^+\} \cup \{a(Y, d') : a \in O^-\} \cup \{Push(X, d' - 1), Pop(X, d')\} \cup \{Move\}$

where X ranges over V, d and d' range over $[0, |V|]$ and $[1, |V|]$ respectively, and O^- and O^+ stand for the sets of actions in O that decrement and do not decrement a variable respectively, and the variable Y in $a(Y, d')$ ranges among the variables decremented by the action a in Q. Preconditions and effects of the actions in O' are described above in the text.

8. Properties

Clearly, the reduction $P = T(Q)$ can be computed in polynomial time:

Theorem 22. *Let $Q = \langle F, V, I, O, G\rangle$ be a QNP. The reduction $P = T(Q)$ can be computed in time that is polynomial in the size of Q.*

Proof. Let $n = |F| + |V|$ be the number of variables, propositional or numerical, in $P = T(Q)$. P has $1 + n$ counters of capacity $1 + 2^n$, each one requiring $1 + n$ bits: the counter $c(d)$ is encoded in binary with bits $c(d, i)$, $i \in [0, n]$. P also has n atoms of form $depth(d)$, $|V| = O(n)$ atoms of form $in(X)$, and $n|V| = O(n^2)$ atoms of form $index(X, d)$. Therefore, P has $O(|F| + n^2) = O(n^2)$ propositional variables.

P has $|V|^2 = O(n^2)$ push actions. Since $Push(X, d)$ has precondition $c(d) < Max$ and effect $c(d) := c(d) + 1$, it gets compiled into n actions of the form $Push(X, d, i)$, $i \in [0, n - 1]$, where precondition $c(d) < Max$ is expressed as $\{\neg c(d, i)\} \cup \{c(d, j) : j \in [0, i - 1]\}$, and effect $c(d) := c(d) + 1$ is expressed as $\{c(d, i)\} \cup \{\neg c(d, j) : j \in [0, i - 1]\}$. The pop actions do not modify counters, so there are $O(n^2)$ of them. The $Move$ action increments the counter c_T and then, like for $Push(X, d)$, it gets compiled into n different actions. Actions a in Q that do not decrement variables are translated into actions a in P. Actions a that decrement a variable get translated into actions $a(X, d)$; there are $O(n^2)$ such actions $a(X, d)$ in P for each such action a in Q.

In total, P has $O(n^2)$ propositional variables and $O(|O|n^2 + n^3)$ actions, where the cubic term accounts for the $Push(X, d, i)$ actions. These numbers (polynomially) bound the size of P. It is clear that producing each action in P is straightforward and can be done in polynomial time. □

The second direct property of the translation is that due to the use of the counters, all strong cyclic policies π for P must terminate:

Theorem 23. *Let Q be a QNP. Any strong cyclic policy π for $P = T(Q)$ is P-terminating.*

Proof. Let π be a strong cyclic policy for P and let $\tau = \bar{s}_0, \ldots, [\bar{s}_i, \ldots, \bar{s}_m]^*$ be an infinite π-trajectory. We need to show that there is some variable X that is decreased in one of these states and increased in none. Clearly, $\pi(\bar{s})$ for some $\bar{s}$ in the recurrent set must be a non-deterministic action, and this means it is an action of form $a(X, d)$. The actions $a(X, d)$ require X to be in the stack and then resets all counters $c(d')$ for $d' \geq d$ back to 0.

Let us pick an action $a(X, d)$ in the recurrent states of τ to be one with smallest stack depth d, and let $\bar{s}$ be one of such states where $\pi(\bar{s}) = a(X, d)$. In the state $\bar{s}$, the variable X is in the stack at level d; i.e., $index(X, d)$ is true. We show next that this same atom must be true in all the other recurrent states in τ. Indeed, if there is a recurrent state where $index(X, d)$ is false, it means that there are recurrent states where X is popped from the stack, and others where it is pushed back at level d, as $\bar{s}$ is a recurrent state where $index(X, d)$ holds. Yet, each occurrence of the action $Push(X, d-1)$ needed to make $index(X, d)$ true increases the counter $c(d-1)$ that no action $a(Y, d')$ can reset with $d' < d$, due our choice of the action $a(X, d)$ as one with minimum d. As a result, it has to be the case that X is in the stack at level d in all the recurrent states of τ, and hence no action that increases X is applied while in a recurrent state (since increments of X are disabled when X is in the stack). Then, since there is a recurrent state where X is decremented, the infinite π-trajectory τ is terminating. Therefore, the policy π is P-terminating. □

In order to prove soundness and completeness, we establish a correspondence between the strong cyclic policies of Q and the strong cyclic policies of $P = T(Q)$. The policies cannot be the same, however, as the reduction T, unlike the direct translation T_D, adds extra variables and actions. Indeed, T preserves the atoms p and $X = 0$ from Q, the latter being propositional, but adds boolean variables and actions that ensure that the policies over of $T(Q)$, unlike the policies over $T_D(Q)$, terminate.

Let Q_M be the QNP obtained from the FOND problem $P = T(Q)$ by 1) adding the numerical variables X from Q, 2) replacing the effects $X > 0$ by $Inc(X)$, and the non-deterministic effects $X > 0 \,|\, X = 0$ by $Dec(X)$, and 3) interpreting the preconditions and goal of the form $X = 0$ and $X > 0$ in terms of such variables (i.e., non-propositionally).

Theorem 24. *If π solves the FOND problem $P = T(Q)$, π solves the QNP Q_M.*

Proof. P is the direct translation of Q_M, i.e. $P = T_D(Q_M)$. π is P-terminating by Theorem 23 and strong cyclic for P as it solves it. Therefore, by Theorem 15, π solves Q_M. □

The QNP Q_M can be thought of as the composition of the original QNP Q and with a deterministic model that encodes the state of the stack and counters. From this perspective, the policy π that solves Q_M stands for a *controller* π_M for Q that has an internal memory M comprised of the atoms that encode the stack and counters: actions like $Push(X, d)$, $Pop(X, d)$ and $Move$ only affect the internal memory M of the controller π_M, actions a that do not decrement any variable in Q, only affect the state of Q, while actions $a(X, d)$ affect both the state of Q and the internal memory M. Due to the correspondence between the application of policy π to the QNP Q_M and the application of the controller with memory π_M to the QNP Q, it is then direct that:

Theorem 25 (Soundness)**.** *If policy π solves the FOND problem $P = T(Q)$, the controller π_M solves Q.*

Proof. Each execution of π in Q_M generates a trajectory over M and one over Q. Since π solves Q_M, the latter must be terminating and goal reaching, but then they must be terminating and goal reaching in Q that shares the same goal as Q_M and the same $Dec(X)$ and $Inc(X)$ actions. □

The inverse direction of this theorem is also true, but it does not give us a completeness result. For that, we need to show that a policy π that solves Q determines a policy π' that solves $P = T(Q)$.

8.1 Completeness

We now assume that there is a policy π that solves Q and want to show that there is a policy $\bar{\pi}$ that solves $P = T(Q)$. Since π solves Q, π is Q-terminating and also P'-terminating where $P' = T_D(Q)$ is the direct translation of Q (cf. Theorem 15).

Let $\mathcal{G}$ be the policy graph associated with π in P'. By Theorem 16, Sieve reduces $\mathcal{G}$ to an acyclic graph. For the rest of this section, we assume that Sieve *is run until all edges that are associated with actions that decrement variables are eliminated* rather than stopping as soon as the graph becomes acyclic.[7] As a result, since π solves Q, the resulting acyclic graph has no edge associated with a decrement of a variable. Each edge removed by Sieve can be identified with a variable, and edges are removed in batches by Sieve, each such $batch(C)$ associated with a component C and a variable X chosen by Sieve; i.e., in a given iteration, Sieve chooses a component C and a variable X, and removes all edges $(\bar{s}, \bar{s}')$ from C such that $\pi(\bar{s})$ is a $Dec(X)$ action (cf. Figure 1).

Let us index the top SCCs processed by Sieve in topological order (i.e., if C_i reaches C_j for $j \neq i$, then $i < j$), and let $scc(\bar{s})$ be the index of the (top) component that includes $\bar{s}$ (i.e., the index of the component that includes $\bar{s}$ in the graph $\mathcal{G}$). Sieve decomposes each component C into a collection of *nested* SCCs that result of recursively removing edges from C. For each state $\bar{s}$, let $\mathcal{C}_{\bar{s}} = \{C^j_{\bar{s}}\}_{j \geq 1}$ be the collection of nested SCCs that contain $\bar{s}$; i.e.,

- $C^1_{\bar{s}} = C_{k_1}$ where $k_1 = scc(\bar{s})$ is the index of the (top) component that contains $\bar{s}$, and
- for $j \geq 1$, $C^{j+1}_{\bar{s}} = C_{k_{j+1}}$ where $\bar{s}$ is in the component $C_{k_{j+1}}$ of the graph that results when all edges in $\bigcup\{batch(C_{k_i}) : i \in [1, j]\}$ have been removed by Sieve.

For each state $\bar{s}$, let $stack(\bar{s})$ be the sequence of variables chosen by Sieve for each component in $\mathcal{C}_{\bar{s}}$. Observe that such sequence contains no repetitions since once Sieve chooses variable X for a component C, the same X cannot be chosen later for another component C' contained in C. Also, if the action $\pi(\bar{s})$ is a $Dec(X)$ action for variable X, then $stack(\bar{s})$ contains X by the assumption that Sieve is run until all edges associated with decrements of variables are eliminated.

We compare the stack α and $stack(\bar{s})$, and say that $\alpha = X_1 \cdots X_n$ is a *prefix* of $stack(\bar{s}) = Z_1 \cdots Z_m$ if the latter can be obtained from the former by pushing variables only; i.e., if $n \leq m$ and $X_i = Z_i$ for $i \in [1, n]$. A property of the Sieve algorithm that we exploit in the completeness proof is the following:

Theorem 26. *Let Q be a QNP and let $P' = T_D(Q)$ be its direct translation. If π solves Q and $\bar{\tau} = \bar{s}_0, \ldots, [\bar{s}_i, \ldots, \bar{s}_m]^*$ is an infinite π-trajectory in P', there is a variable X and a recurrent state $\bar{s}$ such that $\pi(\bar{s})$ is a $Dec(X)$ action, and X is in $stack(\bar{s}')$ for every recurrent state $\bar{s}'$ in $\bar{\tau}$.*

Proof. If π solves Q, π must be P'-terminating. Thus, there must be a variable that is decreased by π in some recurrent state, and increased by π in no recurrent state. At the same time, Sieve is complete and must remove variables (edges) in the policy graph until it becomes acyclic (cf. Theorem 16). Initially, all the recurrent states in $\bar{\tau}$ are in the same component but at one point Sieve removes a variable X and splits the set of recurrent states into different and smaller components. From its definition, this means that X appears in $stack(\bar{s})$ for each recurrent state $\bar{s}$ in

7. Clearly, the modification on the stopping condition for Sieve does not affect its correctness since an acyclic graph remains acyclic when one or more edges are removed, and, on the other hand, if the original Sieve cannot reduced a component, the modified algorithm is not able to reduce it either.

$\bar{\tau}$. Moreover, since the removal of X leaves these states into two or more components, $\pi(\bar{s})$ for one such state must be a $Dec(X)$ action. □

We now define the policy π^* for $P = T(Q)$ that is determined by the policy π that solves Q. In the definition, we use the two functions $scc(\bar{s})$ and $stack(\bar{s})$ defined above in terms of the execution of SIEVE on the policy graph for π on $T_D(Q)$. The states over P are denoted by triplets $\langle \bar{s}, c, \alpha \rangle$ where $\bar{s}$ is the state in $T_D(Q)$, c stands for the state of the counters $c(d)$, $d \in [0, |V|]$, and c_T, and α stands for the state of the stack (given by the atoms $depth(d)$, $in(X)$, $index(X, d)$). The policy π^* for P is defined at triplet $\langle \bar{s}, c, \alpha \rangle$ by

$$\begin{cases} Pop(X,d) & \text{if } c_T < scc(\bar{s}),\ X \text{ is top variable in } \alpha, \text{ and } d = |\alpha|, \text{ else} \\ Move & \text{if } c_T < scc(\bar{s}) \text{ and empty stack, else} \\ Pop(X,d) & \text{if } X \text{ is top variable in } \alpha,\ d = |\alpha|, \text{ and } \alpha \text{ is not a prefix of } stack(\bar{s}), \text{ else} \\ Push(X,d) & \text{if } \alpha X \text{ is a prefix of } stack(\bar{s}) \text{ and } d = |\alpha|, \text{ else} \\ a & \text{if } \pi(\bar{s}) = a \text{ decrements no variable, else} \\ a(X,d) & \text{if } \pi(\bar{s}) = a \text{ decrements } X \text{ at depth } d \text{ in } \alpha \text{ but no other var at depth } d' < d. \end{cases} \quad (15)$$

A first observation is that the policy π^* is defined on every triplet $\langle \bar{s}, c, \alpha \rangle$ such that $\pi(\bar{s})$ is defined. The policy π^* for the FOND problem $P = T(Q)$ on a triplet $\langle \bar{s}, c, \alpha \rangle$ advances the c_T counter until it becomes equal to the index $scc(\bar{s})$ of the SCC in $\mathcal{G}$ that contains the node $\bar{s}$. It then performs pops and pushes until α becomes equal to $stack(\bar{s})$, and finally applies the action a selected by the policy π on Q using the action names a or $a(X, d)$ according to whether a decrements no variable or decrements a variable, which must be in $stack(\bar{s})$ as discussed above. In the latter case, the variable X for the action $a(X, d)$ is the variable X decremented by a that is *deepest in the stack*, at depth d. The completeness result can then be expressed as follows:

Theorem 27 (Completeness)**.** *If π solves the QNP Q, then the policy π^* defined by* (15) *solves the FOND problem $P = T(Q)$.*

Proof. From Theorem 15, π solves Q iff π solves and terminates in $P' = T_D(Q)$. We will show that if π solves and terminates in P', then π^* must solve $P = T(Q)$. Therefore, by forward reasoning, given that π solves Q, then π solves and terminates in P' from which we obtain that π^* solves P.

We need to show that the policy π^* is executable in P, and more precisely that 1) π^* cannot generate non-goal states $\langle \bar{s}, c, \alpha \rangle$ where π^* is not defined or defined but non applicable, and 2) π^* cannot get trapped in a loop that only involves the extra actions $Move$, $Pop(X, d)$ and $Push(X, d)$. These two properties ensure that in any π^*-trajectory over P, if a non-goal state $\langle \bar{s}, c, \alpha \rangle$ is reached, an action a or $a(X, d)$ will be the one changing the component $\bar{s}$ of the state when $\pi(\bar{s}) = a$, and that this will happen after a bounded number of applications of the extra actions $Move$, $Pop(X, d)$ and $Push(X, d)$ that do not change $\bar{s}$. Since the effect of the actions a or $a(X, d)$ on $\bar{s}$ in P is the same as the effect of a on s in P', it follows that π^* will be strong cyclic for P if π is strong cyclic for P'. Alternatively, 1) and 2) ensure that if π^* is executable in P, it generate trajectories over the $\bar{s}$ components that are the same as those obtained by the policy π over P' except for a bounded number of steps where the $\bar{s}$ component in the states $\langle \bar{s}, c, \alpha \rangle$ does not change.

Point 2) is direct. $Move$ increases the counter c_T that no other action decreases. Pushes and pops are applied in order, either to flush out the stack when $c_T < scc(\bar{s})$, or to make $\alpha = stack(\bar{s})$. In the latter case, α is popped until it becomes a prefix of $stack(\bar{s})$ (flushed out in the extreme case), and then pushes take place to make α equal to $stack(\bar{s})$. Hence, no loops that only involve $Move$, $Pop(X, d)$ and $Push(X, d)$ actions are possible.

Point 1) is more subtle. The policy π^* is defined on all triplets $\langle \bar{s}, c, \alpha \rangle$ for which $\bar{s}$ is reachable by π. We first argue, that except for the preconditions on the counters $c(d)$ and c_T, the rest of the preconditions are true for the actions selected by π^*. Observe that every triplet $\langle \bar{s}, c, \alpha \rangle$ reached by π^* is such that $\bar{s}$ is reachable by π, and thus $\pi(\bar{s})$ is defined and applicable in $\bar{s}$. Second, for an action selected by π^*, its easy to see, except for $\neg in(Y)$ when the action is a or $a(X, d)$ and it increments Y, that its preconditions hold. To see that $\neg in(Y)$ also holds, observe that if the actions increments Y, then $stack(\bar{s})$ cannot contain Y; if so, the collection $\mathcal{C}_{\bar{s}}$ of nested components for $\bar{s}$ has a component C that contains a state where Y is decremented while being incremented in $\bar{s}$, thus making Y ineligible by Sieve.

The actions that have preconditions on counters are of type $Push(\Box, d)$ with precondition $c(d) < Max$, and $Move$ with precondition $c_T < Max$. Here, $\Box$ is a placeholder that denotes any variable X in V. For the top counter, $c_T < Max$ always hold since c_T starts at 0, it is only increased to make it equal to $scc(\bar{s})$, and the number of components in $\mathcal{G}$ is less than or equal the number of subsets of states which is less than Max. We are thus left to show $c(d) < Max$ by considering the only type of actions that increase $c(d)$: $Push(\Box, d)$.

For this, we show that π^* cannot generate a trajectory $\tilde{\tau}$ in P that contains a fragment $\tilde{\tau}'$ with $1 + 2^n$ (i.e. Max) actions of the form $Push(\Box, d)$ while no action of the form $a(\Box, d')$, $d' \leq d$, or $Push(\Box, d-1)$ as this would be the only way in which $c(d)$ may grow up to $1 + 2^n$: actions of the form $Push(\Box, d)$ increase $c(d)$ by 1, and the only actions that decreases $c(d)$, back to 0, have the form $a(\Box, d')$ for $d' \leq d$, or $Push(\Box, d-1)$.

Indeed, let $\tilde{\tau}' = \langle \bar{s}_1, c_1, \alpha_1 \rangle, \langle \bar{s}_2, c_2, \alpha_2 \rangle, \ldots$ be such a fragment, and let $1 = i_1 < i_2 < \cdots < i_m$, for $m = 1 + 2^n$, be the indices for the triplets in $\tilde{\tau}'$ on which the policy π^* selects an action of type $Push(\Box, d)$. Observe that between each pair of such indices, there must be one triplet where an action of type a or $a(\Box, d')$ is applied: two pushes at the same stack depth must be mediated by at least one such action.

Let $i_1^* < i_2^* < \cdots$ be the indices such that i_k^* is the first index after i_k where the action selected by π^* is of type a or $a(\Box, d')$, $k \in [1, m]$. Since the total number of states is less than or equal to m, there is some $\bar{s}$ that repeats. Without loss of generality, let us assume that $\bar{s}_{i_1^*} = \bar{s}_{i_m^*}$. The policy π loops in P' on a set $\mathcal{R}$ of recurrent states that includes $\{\bar{s}_{i_k^*} : k \in [1, m]\}$. By Theorem 26, there is a variable X that is decremented by π while looping in $\mathcal{R}$ such that X belongs to each $stack(\bar{s}_{i_k^*})$, $k \in [1, m]$. We choose X to be such variable appearing deepest in the stacks. Therefore, there is index $k \geq 1$ such that $\pi^*(\langle \bar{s}_k, c_k, \alpha_k \rangle) = a(X, d')$ where d' is the depth of X in α_k. Since X also belongs to α_1, it must be the case $d' \leq |\alpha_1| = d$, the latter inequality since $\pi^*(\langle \bar{s}_{i_1}, c_{i_1}, \alpha_{i_1} \rangle)$ is of type $Push(\Box, d)$. This is a contradiction with the assumption that $\tilde{\tau}'$ contains no action of type $a(\Box, d')$ for $d' \leq d$. □

The second reduction from QNPs into FOND problems may be used to compute policies for a given QNP from policies of the resulting FOND. The reduction is a sound and complete mapping. As mentioned above, the resulting QNP policies correspond to controllers that map states $\bar{s}$ and controller states, pairs $\langle c, \alpha \rangle$ that encode the state of the (bounded) counters and stack, into actions. Hence, there is still the question of whether a QNP solvable by such a controller is solvable by a flat policy (as given in Definition 5). In the examples below, the controllers found by the FOND planner over the translation yield flat policies where the selection of actions does not depend on the internal controller state, but more generally the question remains open and beyond the scope of this paper.

9. Examples

Let us illustrate the translation and its solution with a simple example and two variations. The base QNP is $Q_1 = \langle F, V, I, O, G\rangle$ where there are two boolean variables $F = \{p, g\}$, two numerical variables $V = \{n, m\}$, the initial and goal states are $I = \{p, n>0, m>0\}$ and $G = \{g\}$ respectively, and the four actions in O are $a_1 = \langle p, n>0; \neg p, n\!\downarrow\rangle$, $a_2 = \langle \neg p; p\rangle$, and $\mathit{fin}_1 = \langle n=0; g\rangle$ and $\mathit{fin}_2 = \langle m=0; g\rangle$, where the last two actions are used to capture the *disjunctive goal* $n=0 \vee m=0$. The QNP Q_2 is like Q_1 except that $a_2 = \langle \neg p; p, n\!\uparrow\rangle$, while the QNP Q_3 is like Q_1 (and Q_2) except that $a_2 = \langle \neg p, m>0; p, n\!\uparrow, m\!\downarrow\rangle$.

Figure 4 shows the solutions of the FOND problems $P_1 = T(Q_1)$ and $P_3 = T(Q_3)$ for Q_1 and Q_3 that are obtained with the `qnp2fond` translator (see below), and a FOND planner. The QNP $P_2 = T(Q_2)$, on the other hand, has no solution.

In order to understand these results, recall that in the FOND problem $T(Q)$ for any QNP Q, actions that decrement variables may be applied only when some of the decremented variables are in the stack, and actions that increment variables may be applied only when none of the incremented variables is in the stack. In addition, the translation prevents loops where a variable is pushed and popped from the stack, except when the stack contains another variable throughout the loop that is decremented, as otherwise the counters would grow without bounds. With these elements in mind, we can turn to the solutions of Q_1 and Q_3, and the lack of solutions for Q_2.

The solution for Q_1 is direct. The idea is to use action a_1 to decrement n down to zero, and then apply fin_1 to reach the goal. However, a_1 is applicable only if n is in the stack and p is true. Then, n must be pushed into the stack before the first application of a_1, and p must be set to true using a_2 after each use of a_1 as observed in Fig. 4(a).

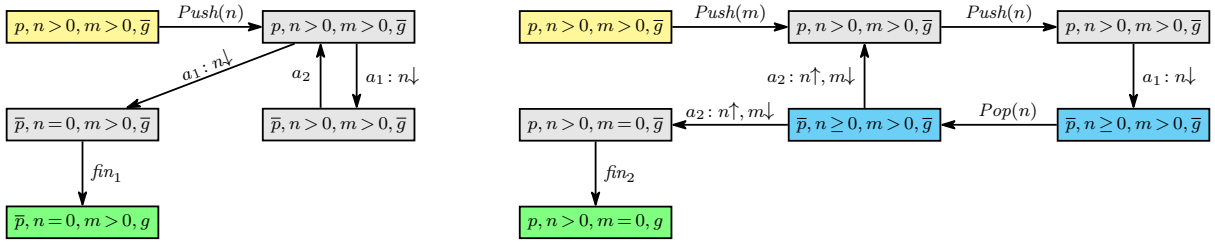
The QNP Q_2 has no solution because the action a_2 that must be used to restore the precondition p of a_1 in the loop increments the variable n as well. Since action a_1 requires n to be in the stack, which prevents an action like a_2 in Q_2 to execute, the only possibility is to pop n from the stack before executing a_2, but then variable n should be pushed and popped from the stack in a loop without the presence of a another variable in the stack that is decremented; a condition that is precluded by the translation, and which is necessary for the loop to terminate.

It is precisely the presence of such extra variable m that is decremented by a_2 and not incremented by any action that makes the QNP Q_3 solvable. While the goal can be achieved by either reaching the condition $n = 0$ or $m = 0$ and then applying the action fin_1 or fin_2 respectively, the solution found by the FOND planner over $T(Q_3)$ focuses on decrementing m down to zero using a_2 instead, and then using fin_2 to reach the goal. The loop involving an unbounded number of pushes/pops of n in the stack is permitted in the solution because the variable m is in the stack and it is decremented in each iteration of the loop. In the translation this means that decrements of m reset the counters associated with the variables like n that are above m in the stack, cf. Fig. 4(b).

Finally, an extra dummy variable like z together with a dummy action $a_3 = \langle z>0; z\!\downarrow\rangle$ would not make the QNP Q_2 solvable either, because while such an action could be used to render a terminating loop involving actions a_1 and a_2, along with a_3, it would move the unsolvability to the subproblem that results when the variable z becomes zero. The policy would need to map states where $z = 0$ holds into goal states, and then, the same obstacles arise.

10. Extensions, Variations, and Memoryless Controllers

For simplicity, QNPs have been defined with certain syntactic restrictions that do not limit their expressive power. In particular, there are no actions with non-deterministic effects on boolean variables, and there are no effects that can map a literal $X=0$ non-deterministically into the



(a) Solution for QNP Q_1

(b) Solution for QNP Q_3

Figure 4: Solutions to the FOND translations $T(Q_1)$ and $T(Q_3)$ of the QNPs Q_1 and Q_3 in the text. Nodes represent states in the translations (i.e., boolean QNP states augmented with stack and counters) but only the QNP part is shown. Edges correspond to actions from Q or actions that manipulate the stack and counters. Edges are annotated with action labels and their effect on the numerical variables. Blue nodes represent multiple QNP states; e.g., the node $\{\overline{p}, n \geq 0, m > 0, \overline{g}\}$ for Q_3 represents the QNP states where $\{\overline{p}, m > 0, \overline{g}\}$ and there is no restriction on the value of n. In both controllers, the initial state is the top leftmost state (in yellow) and the goal is the rightmost state at the bottom (green). The solution for Q_1 decrements n with action a_1 until it becomes zero, and a_1 requires p and thus a_1 is interleaved with a_2 that makes p true. In Q_3, the action a_2 is changed to increment n as well, and to decrement another variable m. The solution for Q_3 found by the solver reaches the goal by decreasing m to zero using action a_2, while using a_1 to restore the preconditions of a_2, and ignoring the variable n. Another solution could be obtained by applying the action fin_1 to the states where $n = 0$ but it would involve more controller states (not shown).

literals $X > 0$ and $X = 0$, as decrements require $X > 0$ as a precondition, and increments yield the outcome $X > 0$. Yet, these two restrictions are not essential and can be bypassed. As we have seen before, non-deterministic effects on boolean variables as found in strong and strong cyclic FOND planning can be obtained in QNPs by using additional numerical variables (Section 6). Likewise, a sequence of two consecutive effects $Inc(X)$ and $Dec(X)$ can be used to emulate the effect of an action that leaves the value of X completely uncertain; i.e., it may increase X, decrease X, or leave the value of X unchanged.

There are also syntactic conditions that when satisfied, make QNPs simpler. For example, if the numerical variables in a QNP Q can be linearly ordered as $X_1, X_2, \ldots$ so that the actions that increase a variable X_i also decrease a variable X_j that appears later in the ordering, then every policy π that solves the direct translation $P = T_D(Q)$ of Q will necessarily solve Q as well, as any such policy will terminate in Q. Indeed, if X_ℓ is the last variable in the ordering that is decreased in a cycle induced by π, the cycle cannot include a different state where the variable X_ℓ is increased, as otherwise, the condition implies that another variable X_j appearing later in the ordering must be decreased in the cycle. For such well-ordered QNPs, the simpler, direct translation T_D is thus both sound and complete.

Finally, recall that a policy π that solves the FOND $P = T(Q)$ obtained from the translation of a QNP Q, defines a policy that can be understood as a memory-extended controller that solves Q using extra boolean variables and actions. Often, however, the policy $\pi(\bar{s}, m)$ obtained from P, where s is the state over Q and m is the memory state, can be projected onto a *memoryless controller* π' for Q which does not use the extra variables or actions. The projection is possible

and immediate when there is no state s in the controller where the actions $\pi(\bar{s}, m)$ and $\pi(\bar{s}, m')$ selected by the policy over two different memory states are associated with different actions in Q. In such a case, all states s can be associated with a single action a from Q (there must be one such action as otherwise π would not map $\bar{s}$ into a goal state), and the memoryless policy π' for Q is then simply $\pi'(\bar{s}) = a$.

11. QNPs and Generalized Planning

QNPs were introduced by Srivastava et al. (2011) as a useful model for planning with loops and for generalized planning (Levesque, 2005; Bonet et al., 2009; Srivastava et al., 2011; Hu & Levesque, 2011; Bonet & Geffner, 2015). In the basic formulation (Hu & De Giacomo, 2011), a generalized planning problem is a collection $\mathcal{Q}$ of planning instances P that share the same set of (ground) actions and the same set state features. The solution of the generalized problem $\mathcal{Q}$ is then a mapping from feature valuations into actions that solves each of the instances in $\mathcal{Q}$. This basic formulation was then extended to domains where the ground actions change from instance to instance, as in most relational domains, like Blocksworld, where the actions are determined by a small number of action schemas and object names. This formulation is achieved by means of QNPs (Bonet & Geffner, 2018) where a single QNP is shown to be capable of representing a suitable abstraction of the concrete instances P involving different ground actions.

A QNP is a *sound abstraction* of a family $\mathcal{Q}$ of concrete problems P from a common domain when the boolean and numerical variables p and n in the QNP accurately represent and track the value changes of certain boolean and numerical state features ϕ_p and ϕ_n in each of the instances. More precisely, a QNP action $\bar{a} = \langle Pre; Eff \rangle$ is sound relative to $\mathcal{Q}$ if in any (reachable) state s over an instance P in $\mathcal{Q}$, if the formula Pre is true in s, with the QNP variables p and n replaced by the state feature functions ϕ_p and ϕ_n, then there is an action in P that induces a state transition (s, s') that agrees with the effects of the abstract action a, once again, with the QNP variables replaced by the corresponding state features.

For example, in Blocksworld, a QNP action $\bar{a} = \langle \neg H, n(x) > 0; H, n(x)\downarrow \rangle$ provides a suitable abstraction of the action of picking up a block from above a designated block x. In this abstraction, the variable H is associated with the boolean state feature ϕ_H that captures when the arm is empty, and the variable $n(x)$ is associated with the numerical state feature ϕ_n that captures the number of blocks above x. The abstract action is sound in the sense that for any state s of a Blocksworld instance P, if $\phi_H(s)$ is false and $\phi_{n(x)}(s) > 0$, there is an action b in P that induces a state transition (s, s') such that $\phi_H(s')$ is true and $\phi_{n(x)}(s') < \phi_{n(x)}(s)$, in agreement with $\bar{a}$. The concrete action b is then said to instantiate the abstract action $\bar{a}$ in the state s of the instance.

In general, if all the QNP actions are sound relative to $\mathcal{Q}$ and suitable conditions apply to the initial and goal conditions of the QNP in relation to $\mathcal{Q}$, any policy π that solves the QNP provides a solution to $\mathcal{Q}$; i.e., the policy π can be applied to any instance P in $\mathcal{Q}$ by interpreting the variables in the QNP in terms of the state features (Bonet & Geffner, 2018). More recently, it has been shown how these QNPs can be learned directly from a PDDL description of the domain and a number of sampled instances and their plans (Bonet et al., 2019), and also how to obtain testable logical conditions to check the soundness of a QNP-based abstraction for an instance P of a PDDL domain description (Bonet et al., 2017). The QNPs used in the experiments below are variations of QNPs learned from samples.

A final question about QNPs for generalized planning is what are the generalized planning problems for which QNPs provide a suitable abstraction and solution method. It turns out that with no restrictions on the state features ϕ_p and ϕ_n that can be abstracted into the QNP, there is indeed, no limit. The solution to any family $\mathcal{Q}$ of planning problems can be expressed compactly

in terms of a single QNP action $\bar{a} = \langle V^* > 0; V^*\downarrow \rangle$ that involves a single numerical variable V^* associated with the feature ϕ_{V^*} that measures the optimal cost of reaching the goal from a given state. The QNP action is sound and just says to move in the direction of the goal. However, the application of this abstract action in a concrete state requires the computation of optimal costs for each successor state, which is in general intractable in the number of problem variables. Thus, a reasonable restriction is that the QNP variables should represent "reasonable" features, and in particular, features that can be computed in polynomial (perhaps linear) time.

12. Implementation: `Qnp2fond`

The reduction from QNPs to FOND problems has been implemented and it can be found in the Github repository `https://github.com/bonetblai/qnp2fond`. The reduction produces a FOND problem without conditional effects which is desirable since some FOND planners do not support them. The reduction may be parametrized in terms of the maximum range of counters and the maximum stack depth, but their default values are those used in the proofs that ensure completeness in general. In some cases, however, the reduction can be made simpler without compromising soundness or completeness. For example, if no numerical variable in the QNP is incremented by an action, the more compact and efficient direct translation T_D is sound and complete. The same holds if the QNP is well-ordered as defined above, where variables may be ordered as $X_1, \ldots, X_n$ such that actions that increment a variable X_i decrement a variable X_j for $i < j$.

On the other hand, if there are actions that increase variables but there are no actions that increase a variable X, there is no need to add push and pop actions for X, nor preconditions for X to be in the stack when decrementing it. This is possible since X may be assumed to be always at the bottom of the stack (as if an implicit action that pushes X has been executed before any other action) and because there is no need to pop X as no action increments it. However, according to the translation, every action that decreases X must reset all stack counters. Interestingly, this and the other simplifications are special cases of a more general simplification that can be applied when the QNP contains a subset S of well-ordered numerical variables, as defined above; indeed, a subset of variables X that are not incremented by any action is such a subset. For such a subset S, one can assume that all variables in S are in the stack and thus simplify the translation by (a) not generating push/pop actions for the variables in S, (b) not adding extra preconditions to actions that decrement a variable in S (such actions already fulfill the requirement that one of its decremented variables must be in the stack), and (c) requiring that actions that decrement any variable in S, reset all the stack counters.[8] When S contains all the numerical variables in the QNP, then there is no need to have a stack and the simplified translation T simply becomes the direct translation T_D.

`Qnp2fond` supports the optimization for variables that are not incremented by any action. The general optimization involving a subset of well ordered variables is not yet implemented since finding

8. Observe that the simplification cannot affect completeness as no action becomes inapplicable since preconditions are removed and counters are potentially reset more often. For soundness, suppose that there is a non-terminating execution and let $\mathcal{R}$ be the set of recurrent actions in such an execution. It cannot be the case that $\mathcal{R}$ contains only actions that affect numerical variables outside S since such actions are not affected by the simplification and thus are subject to the full translation. Hence, there is at least one action in $\mathcal{R}$ that either increments or decrements a variable in S. If the action increments a variable in S, by definition, the action also decrements another variable in S, one that comes later in the ordering. By inductive reasoning, $\mathcal{R}$ contains an action that decrements a "last" variable X_ℓ in $\mathcal{R}$ that is not incremented by any other action in $\mathcal{R}$. Hence, such a variable eventually becomes zero and the action cannot be applied afterwards, contradicting the choice of $\mathcal{R}$.

such a subset and ordering is intractable.[9] There also options for disabling the optimization, and even to force the direct translation whose solutions need to be checked with Sieve. By default, the options are to use the optimization whenever possible, and to use the maximum range for the counters and stack depth that ensure completeness in general.

13. Experiments

We illustrate the performance of the QNP translator and solver over some QNPs that capture abstraction of generalized planning problems. There is no useful baseline for evaluating the use of the translator in combination with FOND planners. The only other complete QNP planner would result from translating QNP problems into LTL synthesis tasks but the comparison would be unfair because LTL synthesis is computationally harder than QNP planning. There is also no complete generate-and-test QNP planner reported, which would have to generate the strong cyclic policies of the direct FOND translation, one by one, while checking them for termination.

In the examples, the resulting FOND problems $T(Q)$ are solved with FOND-SAT (Geffner & Geffner, 2018), a general SAT-based FOND planner that is available at `https://github.com/tomsons22/FOND-SAT`. This planner calls a SAT solver multiple times.[10] The SAT solver used is Glucose 4.1 (Audemard & Simon, 2009) which builds on Minisat (Een & Sorensson, 2004). FOND-SAT solves a FOND problem by constructing a compact controller where each node represents one or more states. When depicting the solutions found, controller nodes that represent more than one state are shown in blue, while the nodes that correspond to the initial and goal QNP states are shown in yellow and green respectively.

13.1 Clearing a Block

A general plan for clearing a given block x in a Blocksworld instance can be obtained by solving the following abstraction expressed as the QNP problem $Q = \langle F, V, I, O, G\rangle$ where $F = \{H\}$ contains a boolean variable H that represents if a block is being held, $V = \{n\}$ contains a numerical variable that counts the number of blocks above x, the initial situation $I = \{\neg H, n > 0\}$ assumes that block x is not clear and that gripper is empty, and the goal situation $G = \{n = 0\}$ expresses that there are no blocks on top of x. There are four actions in O:[11]

- *Putaway* $= \langle H; \neg H\rangle$ to put the block being held on the table or on a block not above x,
- *Pick-above-x* $= \langle \neg H, n > 0; H, n\downarrow\rangle$ to pick the top block above x,
- *Put-above-x* $= \langle H; \neg H, n\uparrow\rangle$ to put the block being held on the top block above x, and
- *Pick-other* $= \langle \neg H; H\rangle$ to pick a block not above x.

`Qnp2fond` translates the QNP Q into the FOND problem $P = T(Q)$ that has 20 atoms and 16 actions in less than 0.01 seconds. FOND-SAT solves P in 0.08 seconds after 3 calls to the SAT solver that require 0.01 seconds in total. The solution produced by FOND-SAT is the controller

9. A subset S and ordering can be found by solving a simple SAT theory, while a maximum-size subset and ordering can be found with a weighted-max SAT solver, or by doing multiple calls to a SAT solver.
10. In each call to the SAT solver, FOND-SAT tries to find a solution (controller) with a given number of states (budget). If no controller is found, the budget is increased by 1, and repeat until one is found.
11. From the point of view of generalized planning, the last QNP action, *Pick-other* $= \langle \neg H; H\rangle$ is not sound in the Blocksworld domain because on the states where there is a single tower and the gripper is empty, the abstract action is applicable yet no concrete action corresponds to it. The obtained policy however is sound as it never prescribes the action *Pick-other*.

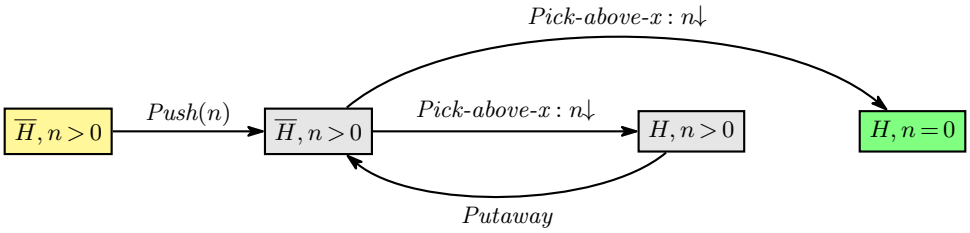


Figure 5: Solution of the FOND translation for the QNP Q for clearing a block x. Nodes represent states in the translation (i.e., QNP states augmented with stack and counters) but only the QNP part is shown. Edges correspond to actions from Q or actions that manipulate the stack and counters. Edges are annotated with action labels and their effect on the numerical variables. The initial state is the leftmost state and the goal is the rightmost one. The policy is strong cyclic and terminating.

shown in Figure 5 which depicts actions from the QNP as well as the added actions to manipulate the stack. The resulting policy is a finite-state controller that can be converted into the memoryless policy π' for the QNP that picks a block above x when the gripper is free, and puts away the block being held otherwise.

13.2 Placing a Block on Top of Another

A general plan for placing block x on top of block y may be obtained by solving a suitable QNP. For simplicity, we only consider the case when the blocks x and y are initially in *different towers* and remains so on the non-goal reachable states (i.e., there are no actions to put blocks above x or y except *Put-x-on-y* that achieves the goal). In this case, the QNP $Q = \langle F, V, I, O, G\rangle$ has boolean and numerical variables $F = \{E, X, D\}$ and $V = \{n, m\}$ that represent whether the gripper is empty (E) or holding the block x (X), or if the goal has been achieved (D), while the numerical variables n and m count the number of blocks above x and y respectively. The initial state $I = \{E, \neg X, \neg D, n>0, m>0\}$ describes a configuration where no block is being held, there are blocks above x and above y, but x and y are in different towers; the goal is simply $G = \{D\}$. The QNP has six different actions:[12]

- *Pick-x* $= \langle E, n=0; \neg E, X\rangle$ to pick block x,
- *Pick-above-x* $= \langle E, n>0; \neg E, n{\downarrow}\rangle$ to pick the topmost block that is above x,
- *Pick-above-y* $= \langle E, m>0; \neg E, m{\downarrow}\rangle$ to pick the topmost block that is above y,
- *Putaside* $= \langle \neg E, \neg X; E\rangle$ to put aside (not above x or y) the block being held,
- *Put-x-aside* $= \langle \neg E, X; E, \neg X\rangle$ to put aside the block x (being held), and
- *Put-x-on-y* $= \langle \neg E, X, m=0; E, \neg X, D, m{\uparrow}\rangle$ to put x on y.

12. One reason for why this particular QNP is not suitable for dealing with states where the blocks x and y are in the same tower, is that the *Pick-above-x* and *Pick-above-y* actions are not sound relative to the intended features ϕ_n and ϕ_m that the variables n and m are aimed to track. A needed *Pick-above-x-and-y* action, for example, will decrement the two variables n and m.

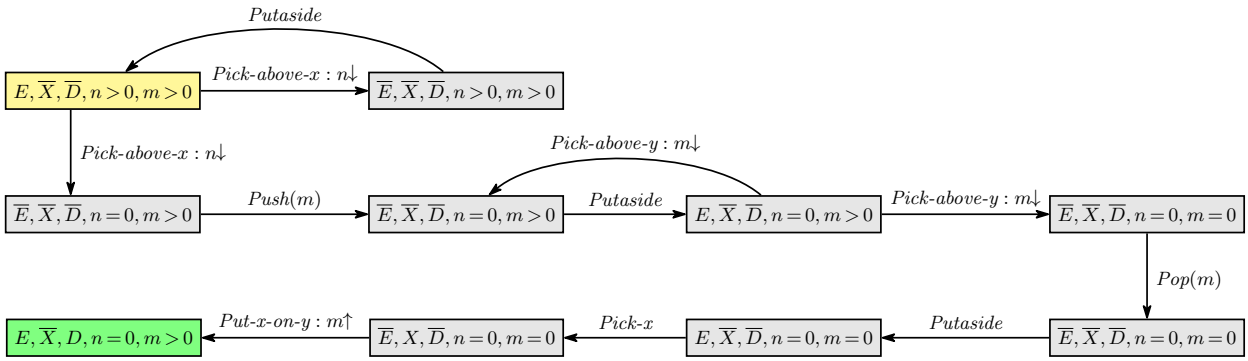


Figure 6: Solution of the FOND translation for the QNP Q for placing a block x on top of another block y. Nodes represent states in the translation (i.e., QNP states augmented with stack and counters) but only the QNP part is shown. Edges correspond to actions from Q or actions that manipulate the stack and counters. Edges are annotated with action labels and their effect on the numerical variables. The initial state is the top leftmost state and the goal is the leftmost one at the bottom. The policy is strong cyclic and terminating.

Since Q has no action to increment n, `qnp2fond` generates in less than 0.01 seconds a simplified FOND problem $P = T(Q)$ that has 47 atoms and 35 actions, 42 atoms for encoding the counters and the stack, and 27 actions that manipulate the stack and the top counter. FOND-SAT finds the solution shown in Figure 6 in 2.55 seconds; it makes 9 calls to the SAT solver that require 0.31 seconds in total. The resulting controller also defines a memoryless policy.

13.3 Gripper

The task involves a robot with grippers whose goal is to move a number of balls from one room into a target room. Each gripper may carry one ball at a time. An abstraction for generalized plan may be obtained with a QNP $Q = \langle F, V, I, O, G\rangle$ that involves one boolean feature T that indicates whether the robot is in the target room, and three numerical features that count the number of balls still to be moved (b), the number of balls being carried (c), and the number of empty grippers (g). The initial state $I = \{T, b>0, c=0, g>0\}$ places the robot at the target room, carrying no balls, and with some balls in the other room. The goal description is simply $G = \{c=0, b=0\}$ saying that the number of balls being carried and the number of balls in the other room are both zero. The set of (abstract) actions in Q is:

– *Drop-at-source* $= \langle \neg T, c>0; b\uparrow, c\downarrow, g\uparrow\rangle$ to drop balls in the other room,

– *Drop-at-target* $= \langle T, c>0; c\downarrow, g\uparrow\rangle$ to drop balls in the target room,

– *Pick-at-source* $= \langle \neg T, b>0, g>0; b\downarrow, c\uparrow, g\downarrow\rangle$ to pick balls in the other room,

– *Move* $= \langle \neg T; T\rangle$ to move to the target room, and

– *Leave* $= \langle T; \neg T\rangle$ to move to the other room.

It is easy to see that this abstraction captures instances involving any number of balls and robots with any positive number of grippers.

The translator runs in less than 0.01 seconds and generates a FOND problem $P = T(Q)$ with 54 atoms and 47 actions (50 atoms for encoding the counters and the stack, and 29 actions to

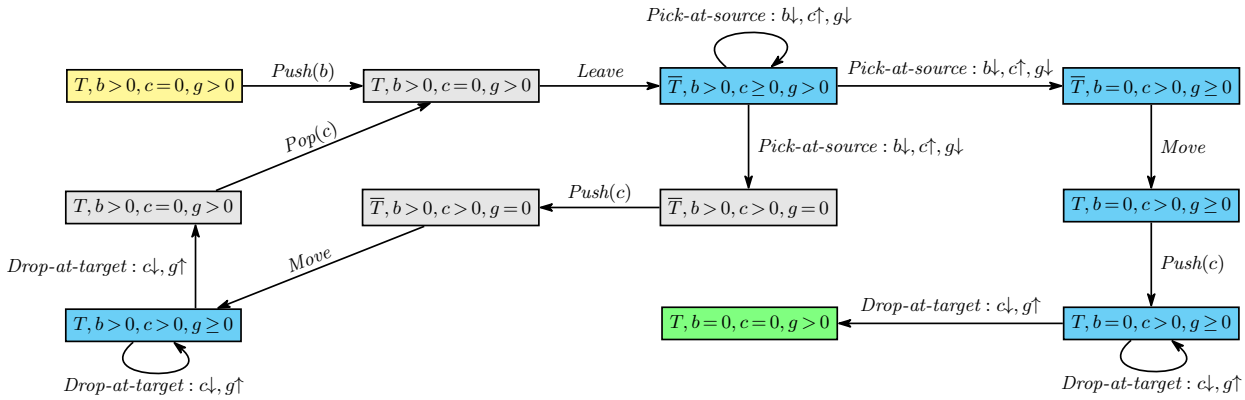


Figure 7: Solution of the FOND translation for the QNP Q for Gripper. Nodes represent states in the translation (i.e., QNP states augmented with stack and counters) but only the QNP part is shown. Blue nodes in the controller represent one or more QNP states; e.g., the node $\{\overline{T}, b>0, c\geq 0, g>0\}$ in the first row represents the QNP states where $\{\overline{T}, b>0, g>0\}$ hold and there is no restriction on the value of c. Edges correspond to actions from Q or actions that manipulate the stack and counters. Edges are annotated with action labels and their effect on the numerical variables. The initial state is the top leftmost state and the goal is the second one at the bottom row. The policy is strong cyclic and terminating because each self loop is terminating, and the outer loop terminates since the variable b decreases by at least one in each iteration, and no action in the plan increases it.

manipulate the stack and move the top counter). FOND-SAT finds the solution shown in Figure 7 in 11.25 seconds; it makes 10 calls to the SAT solver that require 3.00 seconds in total. The resulting controller also defines a memoryless policy.

13.4 Delivery

The last example involves an agent that navigates a grid and whose job is to look for packages and deliver them at a target location subject to the constraint that it can carry one package at a time. The generalized problem consists of all instances with a finite but unbounded grid, and a finite but unbounded number of packages. The generalized problem can be captured with the QNP $Q = \langle F, V, I, O, G\rangle$ that involves one boolean feature H that tells whether the agent is holding a package, and 3 numerical features that measure the distance to the next package (d), the distance to the target location (t), and the number of packages that still need to be delivered (p). The initial state $I = \{\neg H, d>0, t>0, p>0\}$ corresponds to a state where the agent holds no package and is neither at a package or the target location, while the goal description $G = \{\neg H, p=0\}$ indicates that all packages have been delivered. The QNP Q has five actions:[13]

- $Move = \langle d>0, p>0; d\downarrow, t\uparrow\rangle$ to move towards next package and away from target location,
- $Home = \langle t>0; d\uparrow, t\downarrow\rangle$ to move towards target location and away from next package,
- $Pick = \langle \neg H, d=0; H\rangle$ to pick a package,

13. From the point of view of generalized planning, the QNP actions *Move* and *Home* are not sound over this domain since the agent may move towards the next package without moving away from the target location, and vice versa.

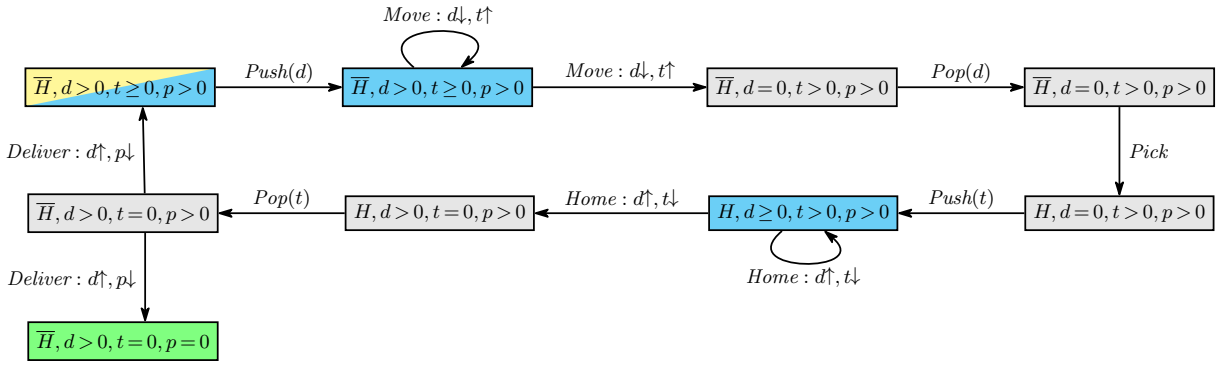


Figure 8: Solution of the FOND translation for the QNP Q for Delivery. Nodes represent states in the translation (i.e., QNP states augmented with stack and counters) but only the QNP part is shown. Blue nodes in the controller represent one or more QNP states; e.g., the top leftmost node $\{\overline{H}, d>0, t\geq 0, p>0\}$ represents the QNP initial state and a similar state except $t=0$. Edges correspond to actions from Q or actions that manipulate the stack and counters. Edges are annotated with action labels and their effect on numerical variables. The initial state is the top leftmost state and the goal is the one at the bottom row. The policy is strong cyclic and terminating because each self loop is terminating, and the outer loop terminates since the variable p decreases by one in each iteration, and no action in the plan increases it.

– $\mathit{Drop} = \langle H, t>0; \neg H\rangle$ to drop a package not in target location, and

– $\mathit{Deliver} = \langle H, t=0, p>0; \neg H, d{\uparrow}, p{\downarrow}\rangle$ to deliver a package in target location.

Since the variable p is not incremented by any action, the translator generates a simplified FOND problem $P = T(Q)$, in less than 0.01 seconds, that has 54 atoms and 40 actions (50 atoms for encoding the counters and the stack, and 29 actions to manipulate the stack and move the top counter). FOND-SAT finds the solution shown in Figure 8 in 2.99 seconds; it makes 8 calls to the SAT solver that require 0.30 seconds in total. As in previous examples, the resulting controller defines a memoryless policy.

14. Related Work

QNPs have been introduced as a decidable planning model able to account for plans with loops (Srivastava et al., 2011, 2015). In addition, by defining the boolean and numerical variables of QNPs as suitable general boolean and numerical features over a given domain, it has been shown that QNPs can be used to express abstract models for generalized planning, in particular when the ground actions change from instance to instance (Bonet & Geffner, 2018). More recently, it has been shown that these QNP abstractions can be learned automatically from a given planning domain and sampled plans (Bonet et al., 2019). QNPs thus provide a convenient language for a **model-based approach** to the computation of general plans where such plans are derived from a (QNP) planning model. If the model is sound, the general plans are guaranteed to be correct (Bonet & Geffner, 2018; Bonet et al., 2019). This is in contrast with the more common **inductive** or **learning-based approaches** where plans computed to solve a few sampled instances are assumed to generalize to other instances by virtue of the compact form of the plans (Khardon, 1999; Martin

& Geffner, 2004; Fern et al., 2004). These learning approaches do not construct or solve a suitable abstraction of the problems as expressed by QNPs. Inductive approaches have been used recently to learn general plans in the form of finite-state controllers (Bonet et al., 2009; Hu & De Giacomo, 2013), finite programs (Segovia-Aguas et al., 2016), and deep neural nets learned in a supervised manner (Toyer et al., 2018; Bueno et al., 2019; Issakkimuthu et al., 2018; Bajpai et al., 2018). A key difference between learning-based and model-based approaches is that the correctness of the latter follows from the soundness of the model. Deep reinforcement learning methods have also been used recently for computing generalized plans with no supervision (Groshev et al., 2018; Sukhbaatar et al., 2015), yet by not using first-order symbolic representations, they have difficulties in dealing with relational domains that involve objects and relations (Garnelo & Shanahan, 2019). Forms of generalized planning have also been formulated using first-order logic (Srivastava et al., 2011; Illanes & McIlraith, 2019), and general plans over finite horizons have been derived using first-order regression as well (Boutilier et al., 2001; Wang et al., 2008; Van Otterlo, 2012; Sanner & Boutilier, 2009). The use of QNPs for expressing (or learning) abstractions for generalized planning problems, combined with the compilation of QNPs into FOND problems, allows us to benefit from the performance of propositional off-the-shelf FOND planners like PRP (Muise et al., 2012), MyND (Bercher & Mattmüller, 2009), and FOND-SAT (Geffner & Geffner, 2018) in order to find generalized plans.

QNP problems can be easily translated into LTL planning problems with FOND domains, reachability goals, and a particular type of trajectory constraints that can be expressed as compact LTL formulas (Bonet et al., 2017). The trajectory constraints use a fragment of LTL (Pnueli, 1977) to express the QNP fairness constraints; namely, that *trajectories where a variable X is decremented an infinite number of times and incremented a finite number of times, are not fair* and can thus be ignored.[14] As a result, QNP planning can be translated quite efficiently (linear time) into LTL synthesis. The translation, however, is not particularly useful computationally, as QNP planning, like FOND planning, is EXP-Complete, while LTL synthesis is 2EXP-Complete (doubly exponential in time) (Pnueli & Rosner, 1989). In LTL planning, i.e., FOND planning with LTL goals and trajectory constraints, the double exponential growth is in the number of variables that appear in such formulas (Camacho et al., 2019; Aminof et al., 2019). Tight complexity bounds for the specific type of LTL trajectory constraints that QNPs convey have not been settled. In any case, such methods need to compute in explicit form the QNP transition system, and thus require exponential space in the total number of variables. This lower bound, that does not consider the LTL formulas associated with the trajectory constrains, already matches the upper bound of the brute-force algorithm that uses Sieve as subroutine (cf. proof of Theorem 17).

15. Conclusions

QNPs are convenient abstract models for generalized planning. In this work we have studied QNPs and placed them on firmer ground by studying their theoretical foundations further. We have also shown that FOND problems can be reduced into QNPs, and vice versa, that QNPs can reduced into FOND problems. Both translations are new and polynomial-time computable, hence establishing that the two models have the same expressive power and the same complexity. The previous, direct translation $T_D(Q)$ for QNPs Q also yields a FOND problem but with fairness assumptions that do not match those underlying strong cyclic FOND planning, and this is why solutions to this translation need to be checked for termination. QNPs can be reduced to LTL synthesis and

14. The LTL formula used by Bonet et al. (2017) is slightly different but logically equivalent, and states that always, if a variable is decreased infinitely often and increased finitely often, it must eventually have value zero. They are equivalent because once the value zero is reached, the variable cannot be decreased without being increased.

planning but these are harder computational tasks. In the future, it would be interesting to study more general types of fairness assumptions and the fragments of LTL that can be handled efficiently with methods similar to the ones developed for QNPs that are based on polynomial-time translations and off-the-shelf FOND planners.

Acknowledgements

We thank the associate editor, Patrik Haslum, and the anonymous reviewers for useful comments that helped us to improve the paper. This work was performed while B. Bonet was at sabbatical leave at Universidad Carlos III de Madrid under a UC3M-Santander Cátedra de Excelencia Award. H. Geffner is also a Guest WASP professor at Linköping University, Sweden, and his work is partially funded by a grant TIN-2015-67959-P from MINECO, Spain, and a grant from the Knut and Alice Wallenberg (KAW) Foundation, Sweden.

References

Aminof, B., De Giacomo, G., Murano, A., & Rubin, S. (2019). Planning under LTL environment specifications. In *Proc. Int. Conf. on Automated Planning and Schedulng (ICAPS)*, pp. 31–39.

Audemard, G., & Simon, L. (2009). Predicting learnt clauses quality in modern SAT solver. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 399–404.

Bajpai, A., Garg, S., & Mausam (2018). Transfer of deep reactive policies for MDP planning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 10988–10998.

Belle, V., & Levesque, H. J. (2016). Foundations for generalized planning in unbounded stochastic domains. In *Proc. Principles of Knowledge Representation and Reasoning (KR)*, pp. 380–389.

Bercher, P., & Mattmüller, R. (2009). Solving non-deterministic planning problems with pattern database heuristics. In *Proc. German Conf. on AI (KI)*, pp. 57–64. Springer.

Bonet, B., Palacios, H., & Geffner, H. (2009). Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proc. Int. Conf. on Automated Planning and Schedulng (ICAPS)*, pp. 34–41.

Bonet, B., De Giacomo, G., Geffner, H., & Rubin, S. (2017). Generalized planning: Non-deterministic abstractions and trajectory constraints. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 873–879.

Bonet, B., Francès, G., & Geffner, H. (2019). Learning features and abstract actions for computing generalized plans. In *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*, pp. 2703–2710.

Bonet, B., Fuentetaja, R., E-Martín, Y., & Borrajo, D. (2017). Guarantees for sound abstractions for generalized planning. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 1566–1573.

Bonet, B., & Geffner, H. (2015). Policies that generalize: Solving many planning problems with the same policy.. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 2798–2804.

Bonet, B., & Geffner, H. (2018). Features, projections, and representation change for generalized planning. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 4667–4673.

Boutilier, C., Reiter, R., & Price, B. (2001). Symbolic dynamic programming for first-order MDPs. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*, Vol. 1, pp. 690–700.

Bueno, T. P., de Barros, L. N., Mauá, D. D., & Sanner, S. (2019). Deep reactive policies for planning in stochastic nonlinear domains. In *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*, Vol. 33, pp. 7530–7537.

Camacho, A., Bienvenu, M., & McIlraith, S. A. (2019). Towards a unified view of ai planning and reactive synthesis. In *Proc. Int. Conf. on Automated Planning and Schedulng (ICAPS)*, pp. 58–67.

Cimatti, A., Roveri, M., & Traverso, P. (1998). Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*, pp. 875–881.

Cimatti, A., Pistore, M., Roveri, M., & Traverso, P. (2003). Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, *147*(1-2), 35–84.

Een, N., & Sorensson, N. (2004). An extensible SAT-solver. *Lecture notes in computer science*, *2919*, 502–518.

Fern, A., Yoon, S., & Givan, R. (2004). Approximate policy iteration with a policy language bias. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 847–854.

Fikes, R., & Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *1*, 27–120.

Garnelo, M., & Shanahan, M. (2019). Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. *Current Opinion in Behavioral Sciences*, *29*, 17–23.

Geffner, H., & Bonet, B. (2013). *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers.

Geffner, T., & Geffner, H. (2018). Compact policies for fully observable non-deterministic planning as sat. In *Proc. Int. Conf. on Automated Planning and Schedulng (ICAPS)*, pp. 88–96.

Ghallab, M., Nau, D., & Traverso, P. (2016). *Automated planning and acting*. Cambridge University Press.

Groshev, E., Goldstein, M., Tamar, A., Srivastava, S., & Abbeel, P. (2018). Learning generalized reactive policies using deep neural networks. In *Proc. Int. Conf. on Automated Planning and Schedulng (ICAPS)*, pp. 408–416.

Helmert, M. (2002). Decidability and undecidability results for planning with numerical state variables. In *Proc. Int. Conf. on Artificial Intelligence Planning Systems (AIPS)*, pp. 44–53.

Hu, Y., & De Giacomo, G. (2013). A generic technique for synthesizing bounded finite-state controllers. In *Proc. Int. Conf. on Automated Planning and Schedulng (ICAPS)*, pp. 109–116.

Hu, Y., & De Giacomo, G. (2011). Generalized planning: Synthesizing plans that work for multiple environments. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 918–923.

Hu, Y., & Levesque, H. J. (2011). A correctness result for reasoning about one-dimensional planning problems.. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 2638–2643.

Illanes, L., & McIlraith, S. A. (2019). Generalized planning via abstraction: arbitrary numbers of objects. In *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*, pp. 7610–7618.

Issakkimuthu, M., Fern, A., & Tadepalli, P. (2018). Training deep reactive policies for probabilistic planning problems. In *Proc. Int. Conf. on Automated Planning and Schedulng (ICAPS)*, pp. 422–430.

Jiménez-Celorrio, S., Segovia-Aguas, J., & Jonsson, A. (2019). A review of generalized planning. *The Knowledge Engineering Review*, *34*.

Khardon, R. (1999). Learning action strategies for planning domains. *Artificial Intelligence*, *113*, 125–148.

Levesque, H. J. (2005). Planning with loops. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 509–515.

Littman, M. L., Goldsmith, J., & Mundhenk, M. (1998). The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, *9*, 1–36.

Martin, M., & Geffner, H. (2004). Learning generalized policies from planning examples using concept languages. *Applied Intelligence*, *20*(1), 9–19.

Muise, C. J., McIlraith, S. A., & Beck, C. (2012). Improved non-deterministic planning by exploiting state relevance. In *Proc. Int. Conf. on Automated Planning and Schedulng (ICAPS)*, pp. 172–180.

Nebel, B. (2000). On the compilability and expressive power of propositional planning. *Journal of Artificial Intelligence Research*, *12*, 271–315.

Pnueli, A. (1977). The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pp. 46–57. IEEE.

Pnueli, A., & Rosner, R. (1989). On the synthesis of an asynchronous reactive module. In *Proc. ICALP*, pp. 652–671.

Rintanen, J. (2004). Complexity of planning with partial observability.. In *Proc. Int. Conf. on Automated Planning and Schedulng (ICAPS)*, pp. 345–354.

Russell, S., & Norvig, P. (2002). *Artificial Intelligence: A Modern Approach*. Prentice Hall. 2nd Edition.

Sanner, S., & Boutilier, C. (2009). Practical solution techniques for first-order MDPs. *Artificial Intelligence*, *173*(5-6), 748–788.

Segovia-Aguas, J., Jiménez-Celorrio, S., , & Jonsson, A. (2016). Generalized planning with procedural domain control knowledge. In *Proc. Int. Conf. on Automated Planning and Schedulng (ICAPS)*, pp. 285–293.

Sipser, M. (2006). *Introduction to Theory of Computation* (2nd edition). Thomson Course Technology, Boston, MA.

Srivastava, S., Immerman, N., & Zilberstein, S. (2011). A new representation and associated algorithms for generalized planning. *Artificial Intelligence*, *175*(2), 615–647.

Srivastava, S., Zilberstein, S., Gupta, A., Abbeel, P., & Russell, S. (2015). Tractability of planning with loops. In *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*, pp. 3393–3401.

Srivastava, S., Zilberstein, S., Immerman, N., & Geffner, H. (2011). Qualitative numeric planning. In *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*, pp. 1010–1016.

Sukhbaatar, S., Szlam, A., Synnaeve, G., Chintala, S., & Fergus, R. (2015). Mazebase: A sandbox for learning from games. *arXiv preprint arXiv:1511.07401*.

Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM journal on computing*, *1*(2), 146–160.

Toyer, S., Trevizan, F., Thiébaux, S., & Xie, L. (2018). Action schema networks: Generalised policies with deep learning. In *Proc. AAAI Conf. on Artificial Intelligence (AAAI)*, pp. 6294–6301.

Van Otterlo, M. (2012). Solving relational and first-order logical markov decision processes: A survey. In Wiering, M., & van Otterlo, M. (Eds.), *Reinforcement Learning*, pp. 253–292. Springer.

Wang, C., Joshi, S., & Khardon, R. (2008). First order decision diagrams for relational MDPs. *Journal of Artificial Intelligence Research*, *31*, 431–472.