Journal of Artificial Intelligence Research 81 (2024) 1703-1759 Submitted 10/2023; published 09/2024

The Goal after Tomorrow: Offline Goal Reasoning with Norms

Pere Pardo PERE.PARDO@UNI.LU
University of Luxembourg
Department of Computer Science
L-4364 Esch-sur-Alzette, Luxembourg

Christian Straßer CHRISTIAN.STRASSER@RUB.DE
Ruhr-Universität Bochum
Philosophie II
D-44780 Bochum, Germany

Abstract

Recent studies have focused on autonomous agents that select their own goals and then select actions to achieve these goals, using online Goal Reasoning (GR). GR agents can revise goals and plans at execution time if unexpected outcomes occur. However, for ethical or legal agent design, even the partial execution of an online plan may result in foreseeable norm violations. To prevent these violations, it is crucial to incorporate GR already at the planning phase. To this end, we design an offline GR system that can harbour normative systems or deontic logics for goal generation. Our main results include a characterization and comparison of the completeness classes for a variety of offline GR planners, and a discussion of the irreducibility of offline GR to pure planning methods.

1. Introduction

Put yourself in the shoes of an autonomous robot: you need first to sensibly select your goals, and then a course of actions to enforce these goals.[1] During this task, bear in mind, your goals and actions will have an impact upon each other: any goal you set will typically motivate new actions, and in turn these actions will likely cause new goals to appear.

At the symbolic level, goal selection tasks are studied in deontic logic and normative systems (Gabbay et al., 2013), while action selection tasks are addressed by automated planning (Ghallab et al., 2004). More recently, a coordinated effort for selecting both plans and goals has been made in the field of goal reasoning (GR) (Aha, 2018). By taking potential goals as primitive elements, online GR methods can effectively be used for unmanned rescue or military operations but, as we will see, do not perform well in norm-regulated societies.

From norms to goals. Rational agents act primarily motivated by internal drives such as needs or desires, but also conform their behaviour to moral, legal and social norms. Prescriptive norms, drives and desires can be conveniently expressed as conditional goals:[2]

$$(condition,\ goal): \quad (p,q) = q \text{ is a goal conditional on } p.$$

1. A third selection task is to reason about (conflicting) beliefs. Beliefs play an important role in BDI agents (Meyer et al., 2015). This additional layer of reasoning is not presently taken into consideration.
2. A prescriptive *norm* is a conditional obligation stemming from an external binding source (law, ethics). Other types of norms, such as permission or constitutive norms, are out of the present scope. For convenience, by *norm* we will refer to any conditional goal —including a conditional desire.

The *goal selection* task is to define one's goals from these norms. A norm (p, q) is *violated* whenever p *but not* q holds, formally $(p \wedge \neg q)$. Two possible readings of a norm (p, q) exist:

$$(p, q) = \begin{cases} \textit{if } p \textit{ is true, then } q \textit{ is a goal} & \text{(factual)} \\ \textit{if } p \textit{ is a goal, then so is } q & \text{(deontic).} \end{cases}$$

Each reading generates a different set of definite goals,[3] as formalized by logical principles for goal detachment called *factual detachment* (FD) and resp. *deontic detachment* (DD):

$$\frac{\begin{array}{ll}\text{[fact]} & p \\ \text{[norm]} & q \text{ if } p\end{array}}{\begin{array}{ll}\text{[goal]} & q\end{array}} \text{(FD)} \qquad \frac{\begin{array}{ll}\text{[goal]} & q \\ \text{[norm]} & r \text{ if } q\end{array}}{\begin{array}{ll}\text{[goal]} & r\end{array}} \text{(DD)}$$

Example 1 (Running example)**.** *You (the agent) desire to meet your friend at a party. You consider it a social obligation to bring snacks at parties.* Let us formalize your desire as an unconditional norm (with the constant $\top = \textit{true}$), and the social obligation as a norm:

$$(\top, \textit{meet}) \qquad\qquad (\textit{party}, \textit{snacks}).$$

In this example, the only goal detached is $\{\textit{meet}\}$, under (FD). (Had your desire been instead $(\top, \textit{party})$, the goals detached by (FD)+(DD) would have been: $\{\textit{party}, \textit{snacks}\}$.)

Often enough, moral or legal norms do conflict with each other or with our desires. For this reason, detachable goals are regarded as *prima facie* goals, from which a selection of *all–things–considered* goals needs to be made.[4] As argued by Van der Torre and Tan (1995), for most conflicts (but not all!) an unfulfilled norm still maintains its normative force and should thus count as violated. The evaluation of a plan should then reflect not only the cost of its execution but also the norm violations the plan will incur in.

From actions to plans. Symbolic propositions not only allow us to express norms and goals. Actions can be represented by pairs of preconditions $\{p, \ldots\}$ and effects $\{q, \ldots\}$ that determine the state reached after an action. The *action selection* task for goal $g = \{q, \ldots, r\}$ (see fn. 3) consists of identifying an action sequence or plan that leads to a state s satisfying the goal: $s \models q \wedge \ldots \wedge r$. A plan for g is built incrementally by a non-deterministic choice over executable actions (forward search); or over instances of *means-ends reasoning* (ME):

$$\frac{\begin{array}{ll}\text{[goal]} & q \\ \text{[action]} & q \ldots \text{ if } p \ldots\end{array}}{\begin{array}{ll}\text{[goals]} & p \ldots\end{array}} \text{(ME)}$$

In either case, conflicts between goals are also found in action selection tasks. These conflicts are due to the practical impossibility, for any plan, to fulfil two goals $\{q_1, q_2\}$, rather than to a logical contradiction between *prima facie* goals, as in $\{q_1, \neg q_1\}$.

Oversubscription planning addresses practical impossibilities with preferences over goals. Deontic logics address contradictory goals with preferences over norms. Very few systems for practical reasoning exist that address both practical and ethical goal dilemmas.

3. The expression '*goal*' here refers to some desirable or obligatory fact, represented by a literal p or $\neg p$. In planning contexts, '*goal*' will often refer to the set of all current such goals, say a set $g = \{p, q, \ldots\}$. The context will make it clear what we mean at each occurrence of the expression '*goal*'.
4. In this sense, norms are more primitive than the goals they detach. Norms can give rise to dispositions to act in some ideal form (of hedonistic, legal or moral nature).

Motivation. Our aim is a formal system that selects goals and actions based on normative reasoning and plan search. Observe that these two selection tasks are mutually dependent: action effects may trigger new goals (FD), and new goals may in turn request further actions to be added to the plan (ME). To this end, we address two research questions:

(Q1) What ways exist to embed norms into planners in order to define GR architectures?

(Q2) Do these agent architectures in practice reduce to pure planning systems?

No particular deontic system is vindicated in this article. As in online GR, the prescriptive component is here a function that assigns goals to states, called the *goal function*. Goal functions encode goals in a simple but inefficient, pointwise manner. For this reason we will focus on goal functions generated by norms under some deontic logic.

Let us review three existing decision pipelines that address question (Q1).

(Naive) A straightforward answer to (Q1) is simply to juxtapose the two selection tasks: formulate goals first and then find a plan for these goals. See the next diagram:

$$\begin{tikzcd}
& \textit{goals} \text{ ⚖} \arrow[dr] & \\
\textit{observe} \text{ 👁} \arrow[ur] & & \textit{plan} \text{ 🧩} \arrow[dl] \\
& \textit{execute} \text{ ⚙} &
\end{tikzcd} \tag{1}$$

After sensing the initial state 👁, a deontic logic endows the agent with a goal ⚖, say $g = \{q, r, \ldots\}$. A classical planner aimed at goal g then provides the agent with a plan 🧩. The agent sequentially executes ⚙ the whole plan to achieve a goal state.

(BDI) Belief-desire-intention models iterate the Naive method (1). BDI agents reason about beliefs 👁 (from observations) and about intentions (from beliefs and desires). After selecting one intention ⚖, the agent picks up a plan 🧩 whose 'precondition' fits the beliefs and its 'effect' fulfils the intention. This plan is executed ⚙. This process is repeated as long as pending or new intentions remain unfulfilled. BDI agents are described by the cycle:

$$\begin{tikzcd}
& \textit{goals} \text{ ⚖} \arrow[dr] & \\
\textit{observe} \text{ 👁} \arrow[ur] & & \textit{plan} \text{ 🧩} \arrow[dl] \\
& \textit{execute} \text{ ⚙} \arrow[ul] &
\end{tikzcd} \tag{2}$$

(OnGR) Online Goal Reasoning agents can revise goals and plans at the execution phase. Such online revisions monitor and deal with false beliefs or unpredictable facts (non-deterministic action effects, adversarial actions) if they present new threats or opportunities. OnGR consists of a cycle of tasks (starting with 👁):

$$\begin{tikzcd}
& \textit{goals} \text{ ⚖} \arrow[dr] & \\
\textit{observe} \text{ 👁} \arrow[ur] & & \textit{plan} \text{ 🧩} \arrow[dl] \\
& \textit{execute}_1 \text{ ⚙} \arrow[ul] &
\end{tikzcd} \tag{3}$$

In this cycle, the *execute*$_1$ task ⚙ enforces just the first action of the plan (or an initial fragment of it) after which the agent observes 👁 the new state. If a discrepancy is found between the expected state and this observation, the agent may adjust the goal ⚖ and then the plan 🧩 accordingly. The OnGR cycle (3) is left on repeat for continual monitoring.

A problem with norms. For norm-based agency, unfortunately, the three methods (1)–(3) fail to accommodate certain action-norm interactions. These failures result in foreseeable norm violations committed by the agent executing the Naive, BDI or OnGR cycle.

Example 2. In the party scenario (Ex. 1), further suppose: *Once you arrive at the party any snack shop will be closed.* Let us represent this detail with a precondition for `buySnacks`:

$$\begin{array}{c|rl}
\textit{norms} & \textit{action} = & (\textit{preconditions}, \textit{effects}) \\
\hline
\left\{ \begin{array}{l} (\top, \textit{meet}) \\ (\textit{party}, \textit{snacks}) \end{array} \right\} & \left\{ \begin{array}{r} \texttt{goParty} = \\ \texttt{buySnacks} = \end{array} \right. & \left. \begin{array}{l} (\{\neg \textit{party}\}, \{\textit{party}, \textit{meet}\}) \\ (\{\neg \textit{party}\}, \{\textit{snacks}\}) \end{array} \right\}
\end{array}$$

The planners in Naive, BDI and OnGR can only provide two possible plans for Ex. 1:

`goParty`	go directly to the party;
`buySnacks.goParty`	buy snacks first and then go to the party.

For our discussion, assume the planners return `goParty`. These methods proceed as follows:

👁 The agent observes the initial state $s_0 = \{\neg \textit{meet}, \neg \textit{party}, \neg \textit{snacks}\}$. 1
⚖ A single goal $\{\textit{meet}\}$ is detached using (FD). Active norm: $(\top,\ \textit{meet})$. 2
🧩 [Assumption] The planner returns `goParty`, a plan from s_0 to a goal state. 3
⚙ The agent executes `goParty`. 4

The Naive method terminates. BDI and OnGR continue:

👁 The agent observes state $s_1 = \{\textit{meet},\ \textit{party}, \neg \textit{snacks}\}$. 5
⚖ A new goal $\{\textit{snacks}\}$ is detached using (FD). Active norm: $(\textit{party}, \textit{snacks})$. 6
🧩 The planner returns *failure*. (No plan for the goal $\{\textit{snacks}\}$ exists from s_1.) 7
⚙ No further action or plan is executed. 8

Result. The final state becomes $\{\ldots, \textit{party}, \neg \textit{snacks}\}$ and so the agent violates the norm $(\textit{party}, \textit{snacks})$.[5] The three methods Naive, BDI and OnGR are thus not 'sound' with respect to norm violations, since a norm-complying plan also exists: `buySnacks.goParty`.

(OffGR) Offline Goal Reasoning. Our analysis of Example 2 is that the revision of plans after any (partial) execution arrives too late. To prevent the norm violation, one must first predict it during the planning phase. To this end, we propose the cycle:

$$\begin{tikzcd}
\textit{observe}\ 👁 \arrow[rr] & & \textit{goals}\ ⚖ \arrow[rr] \arrow[dr] & & \textit{execute}\ ⚙ \\
& \textit{update}\ [⚙] \arrow[ur] & & \textit{plan}\ 🧩 \arrow[ll, bend left] &
\end{tikzcd} \tag{4}$$

5. Observe that it is inessential to the example the use of a complex action `goParty`. In a detailed sequence of primitive actions, only its last move causes *party* and activates the norm (*party*, *snacks*). See (Ghallab et al., 2016) for a formal investigation on the distinction between planning and acting.

where *update* [⚙] simulates an execution ⚙ of the plan under consideration. Note that, strictly speaking, the cycle in (4) only involves goal, planning and update programs. These programs will be formally introduced in Definition 1 as the functions:

$$⚖ = \beta \qquad 🧩 = \varphi \qquad [⚙] = \gamma.$$

Example 3. Under the OffGR cycle (4), steps 1–3 in Example 2 are followed by:

[⚙]	Simulate the execution of `goParty`. Predict the state $\{meet, party, \neg snacks\}$.	4
⚖	A new goal $\{snacks\}$ is detached using (FD). Active norm: $(party, snacks)$.	5
🧩	The planner returns *failure*.	6

Result. This run of OffGR terminates with *failure*, as desired. Further runs on this scenario either repeat steps 1–6, or return the plan `buySnacks.goParty` and dispatch it for execution. An OffGR agent, then, solves the scenario without violating any norm.

Remark 1 (On the elimination of norms)**.** In line with question (Q2), planning practitioners might protest against using Example 2 to prove that norms cannot be addressed by pure planning methods. Any GR problem P defined by norms, they might claim, can be transformed into a planning problem P', say using a compilation $P \longmapsto P'$, such that:

(i) P' only describes a goal (no norms), and
(ii) solving P' means returning a plan that complies with the original norms in P.

The reader is referred to Section 8.4 for a discussion of reductions of this type.

Contributions and overview. One can summarize the above discussion and overview our proposal as follows:

- BDI and Online GR respond to real goals *after* an execution. Offline GR responds to goals predicted after a plan, but it does respond to them *before* its execution.
- Although offline GR planners can be adapted to any goal function, for our completeness results we limit our study to *memoryless* goal functions. That is, the current goal will only depend on the current state, not on the history of previous goals.
- Our definitions for an offline GR system are supported by fundamental results akin to those proved for online GR by Cox (2017). (See Facts 3–4.)

In this article we present an Offline Goal Reasoning system called `OffGR` for the decision pipelines of type (4) and (1). Our main contributions are listed next:

- Six `OffGR` planning algorithms are formally studied and, together with BDI and Online GR methods, compared in terms of performance.
- Each `OffGR` planner is proved equivalent to the computation of least fixed points of some plan revision function.
- We prove the first completeness results for GR algorithms and study their relative power (Figure 1) and the computational cost of generating solutions.

$$\begin{tikzcd}
\texttt{UniClass}^* \arrow[r] \arrow[d, leftrightarrow] & \texttt{Append}^* \arrow[r] & \texttt{Replan}^* \arrow[r] & \texttt{Universal} \\
\beta\texttt{Classical}^* & & & \beta\texttt{Saturate} \arrow[u]
\end{tikzcd}$$

Figure 1: The completeness hierarchy. ($\leftrightarrow$) `OffGR` planners that are complete for the same class of GR problems. ($\rightarrow$) Proper inclusions $\subset$ between completeness classes. Note that β`Saturate` is incomparable ($\not\subseteq, \not\supseteq$) with most `OffGR` planners.

- Using a set of benchmark problems, we identify dilemmas between short- and long-term goal fulfilment as a trade-off between optimality and completeness.
- `OffGR` can be used as a testbed for long-standing debates on detachment principles. With the aim of settling such disputes, `OffGR` can test their performance on benchmark scenarios (deontic puzzles) when one adds actions and intentions.

Finally, the use of norms in offline goal reasoning constitutes an elegant approach to the problems of preference elicitation and representation.

Structure of the paper. Section 2 reviews the literature on agent autonomy based on symbolic representations. Section 3 defines the basic components of our system and Section 4 introduces a general definition for `OffGR` planners. Sections 5–7 study six `OffGR` planners divided in three categories. Section 8 discusses the significance of the `OffGR` system (question Q2), and of the planners under study: their hierarchy and combinability, the local planning function φ, and applications of GR to deontic logic. After concluding in Section 9, an Appendix contains the proofs of our results and explicit algorithms for `OffGR` planners.

Notation. Given two sets A and B, the set A^* contains all finite sequences of elements of A; $\mathcal{P}(A)$ is the power set of A; and $A \setminus B$ denotes their difference. Given two binary relations R and T, their composition is $R \circ T = \{(a, b) : R(a, c) \text{ and } T(c, b) \text{ for some } c\}$; we also let $R^1 = R$ and $R^{n+1} = R \circ R^n$ so that $R^+ = \bigcup_{n \geq 1} R^n$ is the transitive closure of R; and $R^* = R^+ \cup Id$ is its reflexive transitive closure, where Id is the identity relation on the domain of R. $R(a, \cdot) = \{b : R(a, b)\}$ is the set of elements reachable from a in one R-step. For $\alpha \in A$ and $\pi, \pi' \in A^*$, the expressions $\pi.\pi'$ and $\alpha.\pi$ and $\pi.\alpha$ denote the corresponding concatenations.

2. Related Work

As described next, Offline GR overlaps with deontic logic, normative systems and preference-based planning on the selection of goals; with online GR on goal-plan interactions; and finally with BDI approaches on intentions and rational behaviour. A common issue in all these areas is how to aggregate goals or preferences.

Goal Reasoning, surveyed by Aha (2018), studies the continual monitoring of actions and the revision of goals and plans. Earlier work by Knoblock (1995) on interleaving planning and acting was equally motivated by sensing actions and predates modern approaches to GR.

Current work also focuses on online GR, to enable agents to face unexpected opportunities or threats. Partial observability or external events are studied by Jaidee et al. (2011), Klenk et al. (2013) and Paisner et al. (2014) for applications in unmanned aerial or naval operations. Existing planning architectures can incorporate goal deliberation subsystems, as in the work of Shaparau et al. (2006). See also Hawes (2011) for a survey on motivation in intelligent systems along this line. Finally, Cox (2007) proposes an algebraic reduction of online GR to an equation system involving goal, planning and update functions (under an assumption of deterministic actions that equates execution and update). Paisner et al. (2014) extend this work into a cognitive model of human behaviour. Although inspired by this area, the `OffGR` system is aimed at norm-based agency and practical reasoning. Our goal updates are globally defined by deontic inferences upon a state, and so they differ from the atomic goal change operators considered by Aha (2018) and Cox (2017).

Symbolic Planning has also studied extensions of classical planning in line with GR. Partial satisfaction planning (Smith, 2004) studies cost-minimal plans and goal selection for consistent but practically unachievable goal sets. Quantitative approaches also consider utility-cost for net-benefit planning, as in Van den Briel et al. (2004). A hierarchical approach combining HTNs with partial satisfaction can be found in (Behnke et al., 2023). Preference-based planning studies temporal preferences over plan trajectories, see Baier and McIlraith (2008) for an overview. Along the same lines, the planning language PDDL 3 (Haslum et al., 2019) can express goal preferences (as soft goals with violation costs) and metric preferences over plans (based on plan length, cost or trajectory). Planning over temporal goals (Bonassi et al., 2023) does solve our Example 1, but the synthesis of such goals from norms has not been explored so far. As for `OffGR`, preferences over goals derive from conflicting norms under a given logic and are aimed at resolving moral or legal dilemmas; any temporal constraints over goals are addressed by GR itself. Closer to our methodology, Ghallab et al. (2016) provide a planning-based approach to GR. Continual planning, in a similar vein, makes use of plan repair techniques when an agent receives new goals at runtime; see the work by Chien et al. (2000) or Krogt and Weerdt (2005). In the present article we characterize a simple plan repair technique (replanning) for offline GR.

Deontic Logic studies reasoning about obligations and permissions from norms that express conditional obligations or permissions, see Gabbay et al. (2013). Traditionally, deontic logics consist of modal logics based on a betterness relation on states (Chellas, 1980; Hansson, 2013). Monadic modalities $Oq = \textit{it is obligatory that } q$ are studied in (Goble, 2005) to reason about contrary-to-duty (CTD) obligations. CTDs are secondary obligations arising from the violation of a primary norm, such as to apologize after breaking a promise. Dyadic obligations $O(q|p) = q \textit{ is obligatory conditional on } p$ have also been studied in connection with the detachment principles (FD)–(DD) and variants thereof (Hansson, 1969; Loewer & Belzer, 1983; Prakken & Sergot, 1997; Carmo & Jones, 2002, 2013; Straßer, 2011; Parent & van der Torre, 2018). A second group of deontic logics took a non-monotonic turn to address norm conflicts and CTD obligations, see the work by Horty (2012). Non-monotonic systems include: prioritized default logics by Brewka (1989) and Hanssen (2008); defeasible logics by Governatori et al. (2018); adaptive logics in Van de Putte et al. (2019), and argumentation systems as in Pigozzi and van der Torre (2018). While goals can be obligations or desires, deontic logics can still be recruited to generate goal functions. See the study of Doyle et al. (1991) on the relation between desires, obligations and goals.

Normative Systems (Gabbay et al., 2013) study reasoning with norms or about norms. Certain systems derive norms from an initial set of norms, as in input/output logics (Parent & van der Torre, 2018). Closer to our aims are systems that detach goals from norms and facts, such as default logics (Brewka, 1989; Hansen, 2008) or structured argumentation (Pigozzi & van der Torre, 2018). Norm conflicts can be resolved by balancing their priorities (a promise to go out *vs.* saving a drowning child), or because one norm cancels the other (as in legal exceptions); see van der Torre and Tan (1995); arguably, norm violations only take place in the former case, a distinction made in structured argumentation with undercutting attacks (Modgil & Prakken, 2013). Pigozzi and Van der Torre (2018) also consider constitutive norms for the detachment of institutional facts, i.e. claims that an object or event has certain legal or moral status (being a vehicle, constituting a threat).

Machine Ethics studies the moral regulation of artificial agents such as selfdriving cars or smart speakers. Baum (2020) argues that all relevant stakeholders (persons and institutions affected by the agent) should have a say in its regulation. Moor (2006) and Dyrkolbotn et al. (2018) distinguish between implicit and explicit ethical agency, depending on whether immoral choices are ruled out by some external labelling or resp. internal reasoning. These mechanisms may involve learning moral judgements from examples, as in Anderson and Anderson (2014), or be based on some deontic logic or argumentation system, as in Liao et al. (2023). Offline GR can accommodate either mechanism in the goal component.

Preference Elicitation and Representation is the problem of obtaining an agent's preferences and expressing them in compact ways, see Brafman and Domshlak (2009). Traditional representations include goal descriptions, preferences over states and utility functions. Braziunas and Boutilier (2007) propose a regret-based approach under utilities. Haddawy et al. (2003) address the elicitation problem with machine learning. Normative systems offer a compact representation for drives and dispositions, from which qualitative preferences or goals can be inferred. For a quantitative approach, one can assign numerical values (priorities) to norms or soft goals, which can be used to aggregate preferences (Brewka, 2004) or calculate the cost or negative utility of a plan (van den Briel et al., 2004).

Practical Reasoning is the formal study of rational behaviour. Bratman (1991) proposes a planning-based theory of intentions, understood as goals that persist unless extreme circumstances occur. Our memoryless goal functions, in contrast, assume a much weaker notion of intention. Bratman's belief-desire-intention model inspired many logics for reasoning about intentions such as BDI (Rao & Georgeff, 1991; Meyer et al., 2015). BDI implementations rely on planners or plan libraries (Meneguzzi & De Silva, 2015), or plan databases (Shoham, 2009; van Zee et al., 2020). Other BDI architectures incorporate norms (Meneguzzi et al., 2015), obligations (Broersen et al., 2005), or address goal-plan-actions interactions by exploring an AND/OR graph for these interactions (Thangarajah & Padgham, 2011).

3. The `OffGR` System

Let $\mathcal{L}$ be a propositional language over a finite set of atoms $\mathsf{At} = \{p, \ldots\}$ that includes the *true* constant $\top$. (Equivalently, one can see At as a set of ground atoms in a function-free first-order language.) Define the set of literals ℓ by $\mathsf{Lit} = \mathsf{At} \cup \{\neg p : p \in \mathsf{At}\}$. Negation for literals $-\ell$ is expressed by $-p = \neg p$ and $-\neg p = p$. For a set $X \subseteq \mathsf{Lit}$, we also define $-X = \{-\ell : \ell \in X\}$. We will henceforth opt for this negation $-p$ over $\neg p$ in literals.

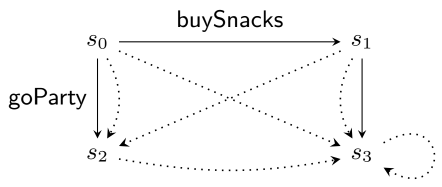


Figure 2: The running example (Examples 1–9). Solid arrows express action transitions. Action labels apply to all arrows with the same direction. Dotted arrows $s_i \dashrightarrow s_j$ express that s_j satisfies all the goals you have at s_i. Observe also the loop at s_3.

A **state** $s = \{\ell, \dots, \ell'\}$ is a consistent and complete set of literals $s \subseteq \mathsf{Lit}$. $S = \{s, \dots\}$ is the finite set of all states over At. The satisfaction relation $\models\ \subseteq S \times \mathcal{L}$ is defined by:

$$\begin{array}{lllll} s \models \top & \Leftrightarrow\ \textit{always} & \qquad & s \models B \wedge C & \Leftrightarrow\ s \models B \text{ and } s \models C \\ s \models p & \Leftrightarrow\ p \in s & & s \models B \vee C & \Leftrightarrow\ s \models B \text{ or } s \models C \\ s \models \neg B & \Leftrightarrow\ s \not\models B & & s \models B \to C & \Leftrightarrow\ s \models B \text{ implies } s \models C. \end{array}$$

The set of states satisfying a given formula $\psi \in \mathcal{L}$ or set $X \subseteq \mathcal{L}$ are defined by:

$$\|\psi\| = \{s \in S : s \models \psi\} \qquad \text{and} \qquad \|X\| = \textstyle\bigcap_{\psi \in X} \|\psi\|.$$

An **action** $\alpha = (name(\alpha), pre(\alpha), eff(\alpha))$ is a tuple such that: $name(\alpha)$ is an identifier for α; $pre(\alpha)$, the precondition for α, is an $\mathcal{L}$-formula or a set of literals; and $eff(\alpha) \subseteq \mathsf{Lit}$ is the set of effects of α. All actions α in our examples will also be introduced as follows:

$$\{\ell, \dots\}\ name(\alpha)\ \{\ell', \dots\} \qquad \text{denoting} \qquad pre(\alpha)\ name(\alpha)\ eff(\alpha).$$

We denote by $A = \{\alpha, \dots\}$ the set of all **available actions**, and use the conventions:

$\cdot\ \pi = \pi[1..n] = (\alpha_1, \dots, \alpha_n)$	a **plan** π, i.e. a finite sequence of actions in A
$\cdot\ A^* = \{\emptyset, \dots, \pi, \dots\}$	the set of **all plans** built from A, incl. the empty plan $\emptyset$
$\cdot\ g \in G = \mathcal{P}(\mathsf{Lit})$	a **goal** $g = \{\ell_1, \dots, \ell_k\}$; the set of **all goals** G
$\cdot\ \Sigma = (S, A)$	a **planning domain** with states S and actions A
$\cdot\ P = (S, A, s_0, g)$	a **classical (planning) problem**[6] P with planning domain (S, A), initial state s_0 and goal g.

Example 4. A planning domain $\Sigma = (S, A)$ for Example 2 (see also Figure 2) consists of:

$$S = \left\{ \begin{array}{c} s_0 = \{-meet, -party, -snacks\} \\ s_1 = \{-meet, -party, snacks\} \\ s_2 = \{meet, party, -snacks\} \\ s_3 = \{meet, party, snacks\} \end{array} \right\} \quad A = \left\{ \begin{array}{c} \{-party\}\ \mathsf{goParty}\ \{party, meet\} \\ \{-party, -snacks\}\ \mathsf{buySnacks}\ \{snacks\} \end{array} \right\}$$

6. The axioms that define classical planning are: finite domains; neither external nor concurrent actions or events; implicit time; deterministic actions and full state observability. See (Ghallab et al., 2004) for details. The same assumptions are made by the `OffGR` system.

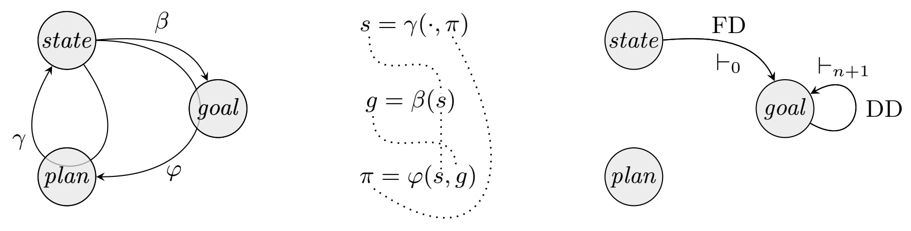


Figure 3: (Left) The (β, φ, γ)-cycle for `OffGR`: goals, plans and updates. (Center) Dependencies among the elements of a GR cycle. (Right) A decomposition of the goal function β that obtains from the detachment principles (Definitions 2–3).

Cox (2017) defines the Online GR cycle (3) with a goal function β, a planning function φ and an execute-observe function γ, and describes equations governing the relations between goals, plans and states. Our formulation of these functions for the Offline GR cycle (4) is simpler, so that β ignores previous goals and φ ignores the plan history so far.[7]

Definition 1 (GR functions, simplified)**.** *The update function γ, goal function β and planning function φ in* `OffGR` *are functions of the following form (see also Figure 3):*

$$\begin{aligned} &\gamma : S \times A^* \to S && \textit{state } \gamma(s,\pi) \textit{ results from executing } \pi \textit{ at } s \\ &\beta : S \to G && \beta(s) \textit{ is the goal } g \textit{ acquired by being at } s \\ &\varphi : S \times G \to A^* && \varphi(s,g) \textit{ is the selected plan for goal } g \textit{ at state } s. \end{aligned}$$

A **GR problem** is a tuple $P = (S, A, s_0, \beta)$ describing: a planning domain (S, A), an initial state $s_0 \in S$ and a goal function β.

Example 5. The GR problem $P = (S, A, s_0, \beta)$ capturing Ex. 2 contains: the planning domain (S, A) from Example 4, the initial state $s_0 = \{-party, -snacks, -meet\}$ and

$$\text{the goal function } \beta = \left(\begin{array}{l} \beta(s_0) = \beta(s_1) = \{meet\} \\ \beta(s_2) = \beta(s_3) = \{meet, snacks\} \end{array} \right).$$

We describe next the three functions γ, β, φ from Definition 1 in detail.

Update function γ. Our update function is that from classical planning in set-theoretic representation (Ghallab et al., 2004). An action $\alpha = (name(\alpha), pre(\alpha), \mathit{eff}(\alpha))$ is called **executable** in a state s if $s \models pre(\alpha)$, and in such case $\gamma(s, \alpha)$ is defined by an update $\diamond$:

$$\gamma(s, \alpha) = s \diamond \mathit{eff}(\alpha) = (s \setminus -\mathit{eff}(\alpha)) \cup \mathit{eff}(\alpha)$$

7. Cox (2017) considers goal functions of the form $\beta : S \times G \to G$. This permits the new goal $g' = \beta(s, g)$ to be partly defined from the previous goal g, although no particular definition is provided to this end. Cox also defines planning functions of the form $\varphi : S \times G \times A^* \to A^*$. That is, the new plan is a function of the previous plan (plus the goal and the state), and so φ acts as a plan revision function. For a comparison, we delegate all plan revision tasks to a global function ψ (Definition 9).

also written $s.\alpha = \gamma(s, \alpha)$. For plans, the update function $\gamma : S \times A^* \to S$ is defined by:

$$\gamma(s, \emptyset) = s \qquad \text{and} \qquad \gamma(s, \alpha.\pi) = \gamma\left(\gamma(s, \alpha), \pi\right).$$

The **action transition relation** $R \subseteq S \times S$ in a planning domain (S, A) is defined as:

$$R = \bigcup_{\alpha \in A} \{(s, \gamma(s, \alpha)) : s \models pre(\alpha)\}.$$

We also write $s \xrightarrow{\alpha} s.\alpha$ and $s \xrightarrow{\pi} s.\pi = \gamma(s, \pi)$ for actions and resp. plan transitions. Given a GR problem $P = (S, A, s_0, \beta)$, we abbreviate the set of **reachable states** as $S^* = R^*(s_0, \cdot)$. Finally, the **trace** of a plan π on state s is the sequence of all states visited during its execution:

$$\widehat{\gamma}(s, \pi) = \langle s, s.\pi[1], \ldots, s.\pi[1..n])\rangle$$

(Note that we will also use $\widehat{\gamma}(s, \pi)$ for the set, rather than sequence, of these states.)

Goal function β. The value of $\beta(s)$ expresses the *all–things–considered* goals: if s is the current state, $\beta(s)$ lists the agent's current goals, and otherwise it describes the goals the agent would have at a possible future state s.

A **goal function** is formally defined as an arbitrary function $\beta : S \to G$ assigning a goal $\beta(s) = \{C, \ldots\} \in G$ to each state s. We also define the **goal relation** $R_\beta \subseteq S \times S$ by:

$$R_\beta(s, \cdot) = \|\beta(s)\| \tag{5}$$

and call any state $s' \in R_\beta(s, \cdot)$ a **goal state** for s. (A goal relation is displayed in Figure 2.) A goal function β is **consistent** if each set $\beta(s)$ is consistent. Observe that β is consistent iff the relation R_β is serial, i.e. there always exists a desirable or goal state $s' \in R_\beta(s, \cdot)$.[8]

Goal functions can be generated in three ways: by manual labour, by a computational implementation of an ethical or legal theory, or from a normative system. We opt for the latter and use norms to represent the agent's drives, ethical claims and legal codes.

A **norm** is a pair (B, C) of conjunctions of literals $(\ell_1 \wedge \ldots \wedge \ell_k, \ell'_1 \wedge \ldots \wedge \ell'_m)$.[9] We indistinctly see a norm as a pair of sets of literals $(\{\ell_1, \ldots, \ell_k\}, \{\ell'_1, \ldots, \ell'_m\})$, and so let $\mathcal{N} \subseteq \mathcal{P}(\mathsf{Lit}) \times \mathcal{P}(\mathsf{Lit})$ be the set of **all norms**. An unconditional norm is of the form $(\top, C)$. We denote by $\mathcal{G} \subseteq \mathcal{N}$ the set of norms of an agent. The goals this agent can detach from its norms $\mathcal{G}$ depend on the detachment principles this agent adopts.

Definition 2 (Detachable goals)**.** *Let $\mathcal{G} \subseteq \mathcal{N}$ be a set of norms and let s be a state. We inductively define the goals C detachable at s (from $\mathcal{G}$) as follows:*

$$\begin{array}{llll} s \vdash_0 & C & \textit{iff} & \textit{there is a norm } (B, C) \in \mathcal{G} \textit{ such that } s \models B \qquad \text{(FD)} \\ s \vdash_{n+1} & C & \textit{iff} & \textit{there is } (\ell_1 \wedge \ldots \wedge \ell_k, C) \in \mathcal{G} \textit{ such that } s \vdash_{\leq n} \ell_1, \ \ldots, \ s \vdash_{\leq n} \ell_k \qquad \text{(DD)} \end{array}$$

where $\vdash_{\leq n} = \bigcup_{k \leq n} \vdash_k$*. We also define* $\vdash = \bigcup_n \vdash_n$ *as the logic based on* (FD)+(DD).

8. *Seriality* expresses a consistency condition also for deontic modal logics based on partial orders. Unlike partial orders, though, goal relations R_β are in general not transitive, and neither reflexive nor irreflexive. Hence, goal relations differ from both non-strict $\preceq$ and strict $\prec$ preferences over states.
9. The language of norms (B, C) can be extended with disjunctive formulas, e.g. $(B \vee \ldots \vee B', C \vee \ldots \vee C')$, with a caveat. *Being obligatory* is not closed under classical consequence, so that inferences of the form *if p is obligatory, so is $p \vee q$* are rejected by many contemporary systems (Ross, 1941). Deontic paradoxes involving disjunction (e.g. Ross paradox) are not addressed in the present article.

Using $\vdash_0$ or $\vdash$, one can just define each value $\beta(s)$ as the set of all goals detachable at s. This goal function β reduces *all–things–considered* goals to *prima facie* goals, and so for typical sets $\mathcal{G}$ of norms, β will be inconsistent (see fn. 17 or Section 8.5). Still, for most of our our examples, goal functions defined by $\vdash_0$ and $\vdash$ will suffice.

Definition 3 (Goal functions β_g, β_0 and β_ω)**.** *We define three types of goal function:*

- $\beta_g(s) = g$ *constant functions* $\beta_g : S \to \{g\}$ *defined by a goal* $g \in G$
- $\beta_0(s) = \{C \in \mathcal{L} : s \vdash_0 C\}$ *the* (FD) *goal function*
- $\beta_\omega(s) = \{C \in \mathcal{L} : s \vdash C\}$ *the* (FD)+(DD) *goal function.*

A **deontic logic**, for the present purposes,[10] is a relation $\mid\!\sim\ \subseteq (S \times \mathcal{P}(\mathcal{N})) \times \mathcal{L}$ satisfying $\mid\!\sim\ \subseteq\ \vdash$. Using an infix notation for deontic logics $\mid\!\sim$, instead of $(s, \mathcal{G}, C) \in\ \mid\!\sim$ we write:

$(s, \mathcal{G}) \mid\!\sim C$ *C is a goal at s based on logic $\mid\!\sim$ and norms $\mathcal{G}$.*

Thus, condition $\mid\!\sim\ \subseteq\ \vdash$ states that all goals are detachable: $(s, \mathcal{G}) \mid\!\sim C \ \Rightarrow\ (s, \mathcal{G}) \vdash C$.[11]

A **normative system** is a pair $(\mid\!\sim, \mathcal{G})$ consisting of a deontic logic $\mid\!\sim$ and a set of norms $\mathcal{G}$. Each normative system $(\mid\!\sim, \mathcal{G})$ produces an **induced goal function** $\beta : S \to G$, namely

$$\beta(s) = \bigcup\{C \subseteq \mathsf{Lit} : (s, \mathcal{G}) \mid\!\sim C\} \tag{6}$$

Goal functions induced by deontic logics $\mid\!\sim$ from the literature are studied in Section 8.5.

Fact 1. *Normative systems $(\mid\!\sim, \mathcal{G})$ are as expressive as goal functions $\beta : S \to G$ when defined over the same set* Lit *of literals.*

Example 6. Recall the norms $\mathcal{G} = \{(\top, \mathit{meet}), (\mathit{party}, \mathit{snacks})\}$ from Ex. 1. The normative system $(\vdash_0, \mathcal{G})$ induces the following goal function β_0 and goal relation $R_{\beta_0}(s, \cdot) = \|\beta_0(s)\|$:

$$\begin{aligned} \|{-\mathit{party}}\| = \{s_0, s_1\} &\xrightarrow{\beta_0} \quad \{\mathit{meet}\} \quad \xrightarrow{\|\cdot\|} \{s_2, s_3\} \\ \|\mathit{party}\| = \{s_2, s_3\} &\xrightarrow{\beta_0} \{\mathit{meet}, \mathit{snacks}\} \xrightarrow{\|\cdot\|} \{s_3\}. \end{aligned}$$

This β_0 is just the function β piecewise defined in Example 5. The goal relation R_{β_0} is also depicted in Figure 2. Its reflexive edge $R_{\beta_0}(s_3, s_3)$ means that state s_3 fulfils the goal at this state: $s_3 \models \beta(s_3)$. Reflexive edges of R_β will define the solution concept in `OffGR`.

Planning function φ. Plans are the third ingredient of the offline GR cycle. As in online GR, classical plans will suffice. A **plan for** $P = (S, A, s, g)$ is a plan $\pi \in A^*$ satisfying:

$$\gamma(s, \pi) \models g \tag{7}$$

and in such case we also say that π **solves** P. Similarly to Cox (2017), we generalize condition (7) to our (β, φ, γ)-cycle. Each call to the planner φ should return, if possible, a

10. Traditionally, a deontic logic is a consequence relation $\models_O\ \subseteq \mathcal{P}(\mathcal{L}_O) \times \mathcal{L}_O$ over a language $\mathcal{L}_O$ with an obligation modality $O(\cdot)$. One can write norms $\mathcal{G}$ and facts s as $\mathcal{L}_O$-formulas and use them as premises to induce a goal function: $\beta(s) = \{C \in \mathcal{L} : s \cup \mathcal{G} \models_O O(C)\}$. In this case, $\beta(s)$ is an infinite set of goals.
11. Another condition that could be imposed to deontic logics $\mid\!\sim$ is *subgoal closure*: if $(B, C) \in \mathcal{G}$ is used for some (DD)-detachment $(s, \mathcal{G}) \mid\!\sim C$, then also $(s, \mathcal{G}) \mid\!\sim B$. This property holds for deontic modal logics.

Algorithm 1 `Classical*` % a depth first search forward planner

Input: a problem (S, A, s_0, g)

Output: a plan π or *failure*

```
1:  if $s_0 \models g$ then
2:      return $\emptyset$
3:  end if
4:  Let $s = s_0$ and $\pi = \emptyset$
5:  while $s \not\models g$ or Choice do                                % calls Algorithm 2
6:      $A' = \{\alpha \in A : s \models pre(\alpha)$ and $\gamma(s,\alpha) \notin \widehat{\gamma}(s_0,\pi)\}$   % never visit a state twice
7:      if $A' = \emptyset$ then
8:          return failure
9:      else
10:         choose $\alpha \in A'$
11:         set $s \leftarrow \gamma(s,\alpha)$    $\pi \leftarrow \pi.\alpha$
12:     end if
13: end while
14: return $\pi$
```

Algorithm 2 `Choice` % a non-deterministic choice

Output: a value `true` or `false`

```
1: choose value $\in \{$true, false$\}$
2: return value
```

plan $\pi = \varphi(s, \beta(s))$ from s to the goal $\beta(s)$. In principle, φ can be any function satisfying the soundness condition:

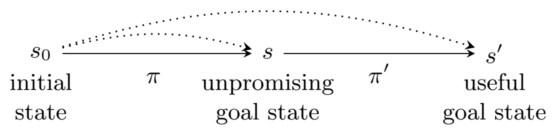
$$\gamma(s, \varphi(s, \beta(s))) \models \beta(s) \tag{8}$$

Equation (8) does indeed capture the minimum requirements of executability and goal fulfilment we wish to impose on φ. So now the question is: what should φ be? Online GR algorithms typically rely on a **classical forward search** planner, here simply called `Classical`, whereas we will mainly consider a slight **variant** of it, that we call `Classical`*. Thus, as the planning domain (S, A) will be clear from the context, we will just write:

$$\varphi(s, g) = \texttt{Classical}^*(S, A, s, g) \tag{9}$$

Algorithm 1 defines this planner. `Classical`* deviates from `Classical` in that it might not stop at the first plan that solves a classical problem $P = (S, A, s_0, g)$. Instead, every time `Classical`$^*(P)$ finds a non-empty plan π for P, it calls Algorithm 2 to choose between two options: (1) to return π or (2) to expand π into another plan for g. (`Classical` is Algorithm 1 after removing the underlined text in line 5 and the now redundant lines 1–3.)

While this variant `Classical`* makes little sense for classical planning or online GR, for offline GR it avoids getting trapped in local maxima, such as s in the next illustration:


$$\begin{tikzcd}
s_0 \arrow[r, "\pi"'] \arrow[r, dotted, bend left] \arrow[rr, dotted, bend left] & s \arrow[r, "\pi'"'] & s' \\
\text{initial state} & \text{unpromising goal state} & \text{useful goal state}
\end{tikzcd}$$

Local maxima are explained in Section 8.2. `Classical`* provides `OffGR` planners with better completeness classes, among other advantages (also explained in Section 8.2).

To ensure termination, Algorithm 1 keeps track of all nodes expanded by the plan (i.e. its trace) to to prevent visiting them twice in an execution of `Classical`$^*(P)$. Because of this, the next completeness result focuses on non-redundant plans. A plan π for $P = (S, A, s_0, g)$ is **non-redundant** if the states in the (sequence) trace $\widehat{\gamma}(s_0, \pi)$ are pairwise different.

Proposition 2. *Let $P = (S, A, s_0, g)$ be a classical problem. (Soundness.) If* `Classical`$^*(P)$ *returns π, then π is a classical plan for P. (Strong completeness.) If $s_0 \not\models g$, then for any non-redundant plan π for P, there is an execution of* `Classical`$^*(P)$ *that returns π. (Termination.) Any execution of* `Classical`$^*(P)$ *terminates after a finite number of steps.*

Observe that strong completeness implies completeness (that is, the planner returns a solution if one exists). For a comparison, `Classical` is complete but not strongly complete.

Fact 3. *The soundness condition $\gamma(s, \varphi(s, \beta(s))) \models \beta(s)$ in (8) holds for classical planning if we interpret $\varphi =$* `Classical` *or $\varphi =$* `Classical`*.

How is φ used in `OffGR`? Recall the cycle *goals* $\rightarrow$ *plan* $\rightarrow$ *update* $\rightarrow$ *goals* in the offline GR cycle (4). Every time one updates the current state (γ), a new goal might exist (β), in which case a new call to the planning function can be made (φ).

Example 7. Let us formalize Example 3. After the initial goal $\beta(s_0) = \{meet\}$, the first invocation of φ returns either:

$$\varphi(s_0, \beta(s_0)) = \texttt{goParty} \quad \text{or} \quad \varphi(s_0, \beta(s_0)) = \texttt{buySnacks.goParty}.$$

- `goParty` leads to state $s_2 = \{party, meet, -snacks\}$ with goal $\beta(s_2) = \{meet, snacks\}$. At this point, a new invocation of φ returns $\varphi(s_2, \beta(s_2)) = failure$.
- `buySnacks.goParty` leads to $s_3 = \{party, meet, snacks\}$, a state that already fulfils the goals it generates. Any further call to φ would be pointless: $\varphi(s_3, \beta(s_3)) = \emptyset$.

At every run of the Offline GR cycle (4), φ acts as a **local function** possibly returning a classical plan. This plan can be seen as a piece of a bigger plan obtained from a succession of calls made to φ. We address next the construction of such a global plan. This global plan will exit the cycle in (4) and so it will be the plan recommended for execution.

4. `OffGR` Planners: General View

As mentioned before, `OffGR` replaces a goal specification g by a goal function β.

Definition 4 (GR problem, simplified)**.** *A* GR problem *is a tuple $P = (S, A, s_0, \beta)$ containing: a planning domain (S, A), an initial state $s_0 \in S$ and a goal function $\beta : S \rightarrow G$. The class of all GR problems is denoted* $\mathbf{P}$.

Definition 5 (`OffGR` planner)**.** *An* `OffGR` planner *is a (non-deterministic) function:*

$$\texttt{Planner}\colon \mathbf{P} \rightarrow A^* \cup \{failure\}$$

that returns for each GR problem $P = (S, A, s_0, \beta)$ either a sequence $\pi \in A^*$ or 'failure'.

Abusing notation, we also denote by `Planner`(P) *the set of possible outputs $\pi \in A^*$ returned by some execution of a non-deterministic algorithm for* `Planner` *with input P.*[12]

This definition does not specify how β is to be used by particular `OffGR` planners. To this end, notice that each goal function β determines a (possibly empty) set of states that are "fixed points" of β, i.e. states with reflexive edges in R_β. Any such fixed point s is called a **solution state**, and so we define the set $\beta^\star \subseteq S$ of solution states (w.r.t. β) as follows:

$$s \in \beta^\star \Leftrightarrow s \models \beta(s) \tag{10}$$

Given a GR problem $P = (S, A, s_0, \beta)$, a **solution** for P is a plan π to a solution state:

$$\gamma(s_0, \pi) \models \beta(\gamma(s_0, \pi)). \tag{11}$$

Fact 4 (Classical to GR)**.** *Each classical problem (S, A, s_0, g) reduces to the GR problem (S, A, s_0, β_g) defined by the constant goal function $\beta_g(s) = g$, for any $s \in S$.*

Example 8. The plan `buySnacks.goParty` from Example 7 is a solution as it leads to a state s_3 satisfying $s_3 \models \beta(s_3)$. The plan `goParty` is not a solution since $s_2 \not\models \beta(s_2)$.

Our solution concept is motivated by the fact that an agent reaching the region $\beta^\star$ needs not act any further. (This does not mean, though, that all solutions are equally good.)

Fact 5. *Let $\varphi =$* `Classical` *or $\varphi =$* `Classical`**. Then, $s \in \beta^\star$ iff $\varphi(s, \beta(s)) = \emptyset$.*

Finding a plan to $\beta^\star$ is thus our condition for exiting the offline GR cycle in (4) and proceed to execute the plan. This will be the ultimate target of all `OffGR` planners, and in fact they will stop at the first solution they encounter. Thus, the **search space** $S' \subseteq S$ will consist of states that can be reached without traversing any (other) solution state:

$$S' = \big\{\gamma(s_0, \pi) \in S : \widehat{\gamma}(s_0, \pi) \cap \beta^\star \subseteq \{\gamma(s_0, \pi)\}\big\}.$$

Definition 6 (Soundness)**.** *An* `OffGR` *planner* `Planner` *is* sound *for a GR problem P iff*

if `Planner`(P) *returns π, then π is a solution for P.*

We also say that `Planner` *is sound for a class $\mathbf{P'} \subseteq \mathbf{P}$ iff it is sound for each $P \in \mathbf{P'}$.*

Definition 7 (Completeness)**.** *An* `OffGR` *planner* `Planner` *is* complete *for some $P \in \mathbf{P}$ iff*

if a solution exists for P, there is an execution of `Planner`(P) *that returns a solution.*

We also say that `Planner` *is complete for a class $\mathbf{P'} \subseteq \mathbf{P}$ if it is complete for each $P \in \mathbf{P'}$. The completeness class of* `Planner` *is the $\subseteq$-maximal class $\mathbf{P'} \subseteq \mathbf{P}$ for which it is complete.*

Finally, we just say that `Planner` is sound (complete) if it is sound (resp. complete) for $\mathbf{P}$.

Definition 8 (Termination)**.** *An* `OffGR` *planner* `Planner` *has the* termination *property if for any GR problem P, any execution of* `Planner`(P) *terminates after finitely-many steps.*

12. `Classical`$^*(P)$ and `Classical`(P) (and `Planner`$_{\triangleright T}(P)$ in Section 8.4) will analogously denote the corresponding sets of possible plans returned by these algorithms for a given classical problem $P = (S, A, s_0, g)$.

Algorithm 3 ψ-`Planner` $\quad lfp(\psi(P))$

Input: a function $\psi : A^* \to A^*$; a GR problem $P = (S, A, s_0, \beta)$
Output: a plan π or *failure*.

```
1: Let $\pi = \emptyset$.
2: repeat
3:     $\pi \leftarrow \psi(\pi)$
4: until $\psi(\pi) = \pi$
5: return $\pi$
```

For mathematical convenience, `OffGR` planners will be defined as non-deterministic algorithms. One can make them deterministic as usual, by imposing both a search order on the set A for node expansion and a search strategy. This suffices for the next result. (An analogous result for non-determinstic algorithms can be shown as well under the mild assumption of *computational fairness*. We omit any details for the sake of brevity.)

Fact 6. *If* `Planner` *is sound, complete and satisfies termination (Definitions 6–8), then a determinstic execution of* `Planner`(P) *returns a solution for* P *iff a solution for* P *exists.*

Henceforth, all `OffGR` planners based on `Classical`* will be named with an asterisk (e.g. `Append`*), and without an asterisk if based on `Classical` (e.g. `Universal`). In the next sections we will study `OffGR` planners of both types, built upon different principles: (i) a blind search for the $\beta^\star$ region, (ii) a continual search for goal states, or (iii) an attempt to discover a $\beta^\star$-state before the planning phase. Our planners thus divide into:

(*i*) **α-methods.** Explore the search space, one action α at a time, and check for goal- and/or solution states. Planners: $\{\beta$`Classical`*, `Universal`, `UniClass`$^*\}$.

(*ii*) **β-methods.** Compute the current goal $\beta(s)$. Invoke the φ function towards $\beta(s)$. Repeat until $\beta^\star$ is reached. Planners: $\{$`Append`*, `Replan`$^*\}$.

(*iii*) **β-saturation methods.** Update the current state s with goals $\beta(s)$, until a solution state is found. Invoke φ to this state. Planner: $\{\beta$`Saturate`$\}$.

As a matter of fact, each of these six `OffGR` planners is also equivalent to the search for a fixpoint of a **plan revision** function ψ. That is, a function that maps plans to plans:

$$\psi : A^* \to A^* \qquad \textit{after constructing plan } \pi\textit{, revise it into } \pi' = \psi(\pi).$$

All of our `OffGR` planners will first be introduced this way using Algorithm 3. Their equivalent programs can be found as Algorithms 4–9 in the Appendix.

Definition 9 (ψ function, ψ-plan)**.** *A* ψ function *is a map* $\psi : P \longmapsto \psi(P)$ *from each GR problem* P *to a plan revision function* $\psi(P)$ *just denoted* $\psi : A^* \to A^*$*. (Here the set* A^* *includes 'failure' as well.) A* ψ-plan *for* P *is any fixed point* $\pi = \psi(\pi)$ *of this function.*[13]

Algorithm 3 takes as inputs a ψ function and a GR problem and returns $lfp(\psi(P)) = \pi$, a least fixed point (*lfp*) of ψ. This way, once we fix ψ, we get an `OffGR` planner (Def. 5):

$$lfp(\psi(\cdot))\colon P \longmapsto \pi \tag{12}$$

13. We also stipulate that $\psi(\textit{failure}) = \textit{failure}$ but this "fixed point" does not count as a ψ-plan.

In general, `OffGR` planners will use the input $P = (S, A, s_0, \beta)$ as follows in Equation (12):

(A) the available actions obviously constrain how plans π can be revised into $\psi(\pi)$.
(s_0) the initial state is needed for $\psi(\emptyset)$ to be well-defined, i.e. executable at s_0.
(β) revisions $\pi \longmapsto \psi(\pi)$ may be motivated by the goal after π or the region $\beta^\star$.

Needless to say, we want ψ functions whose outputs for P (the ψ-plans) are solutions for P. For these ψ functions, solving offline GR problems reduces to solving a fixpoint equation $\pi = \psi(\pi)$. As one might expect, the recursive fixpoint equation for online GR (Cox, 2017, Eq. 7) is way more complex than the equations for `OffGR` (see e.g. Defs. 13–14).

For the sake of comparison, we adapt Online GR and BDI algorithms to simplified GR problems (Definition 4), and call the adjusted algorithms `OnlineGR` and `BDI`. Let us observe that the execution step ⚙, essential to these algorithms, is not part of `OffGR` algorithms, as the latter only return plans (Algorithm 3). For a unified treatment, in the examples any final output plan (executed or not) is identified as the plan returned by the algorithm.

5. `OffGR` Planners: α-Methods

We call α-methods those planners that consider one action α at a time —hence the name. β`Classical`* is a formal version of Naive (1). `Universal` is an exhaustive search method focused on the $\beta^\star$ region. And `UniClass`* combines features from these two planners.

5.1 β`Classical`*

Following the naive method (1), β`Classical`* settles for the initial value $\beta(s_0)$ and then invokes $\varphi =$ `Classical`*. No revision of goals or plans occur, and the ψ-function is trivial.

Definition 10 (β`Classical`*). *β`Classical`* is defined as lfp$(\psi(\cdot))$ under the ψ function:*

$$\psi(\pi) = \begin{cases} \varphi(s_0, \beta(s_0)) & \text{if } \pi = \emptyset \\ \pi & \text{otherwise.} \end{cases}$$

Notice that a fixed point $\psi(\pi) = \pi = \psi(\emptyset)$ is always achieved after the first call to φ. (With detail: if $\emptyset$ is a solution, then $\varphi(s_0, \beta(s_0)) = \emptyset$ and so $\psi(\emptyset) = \emptyset$; otherwise, we get $\pi = \varphi(s_0, \beta(s_0)) \neq \emptyset$ and then $\psi(\pi) = \pi$ after a single step.)

Lemma 7. *Let ψ be as in Definition 10 and let β`Classical`* be the planner from Algorithm 4 (in the Appendix). For any GR problem $P = (S, A, s_0, \beta)$,*

$$\mathit{lfp}(\psi(P)) = \texttt{Classical}^*(P') = \beta\texttt{Classical}^*(P)$$

where $P' = (S, A, s_0, \beta(s_0))$ is the transformation of P into a classical problem.

Proposition 8. *β`Classical`* has the termination property but is neither sound nor complete for $\mathbf{P}$. In fact,*

(i) β`Classical`* is sound for P iff $P \in \mathbf{P}' = \{(S, A, s_0, \beta) : \|\beta(s_0)\| \cap R^*(s_0, \cdot) \subseteq \beta^\star\}$.

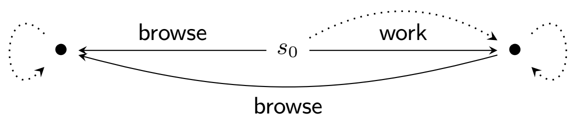


Figure 4: Akrasia, from Example 10.

(ii) β`Classical`* *is complete for P iff if a solution for P exists then a solution to some state in $\|\beta(s_0)\|$ also exists.*

As a consequence, (iii) classical planning is subsumed by β`Classical`**.*

Example 9. Let P be as in Example 6. We verify that β`Classical`* is not sound for P: $\|\beta(s_0)\| \cap S' = \{s_2, s_3\} \not\subseteq \{s_3\} = \beta^\star$ (Proposition 8). More generally, we have:

- $\{\beta$`Classical`*, `OnlineGR`, `BDI`$\}$ return `goParty` or `buySnacks.goParty`.
- all other `OffGR` planners (Algorithms 5–9) return the solution `buySnacks.goParty`.

5.2 `Universal`

This exhaustive search method performs a local search itself (does not call φ) and tests if each state is a solution state, ignoring all previous goals along the path.

Definition 11 (`Universal`)**.** `Universal` *is defined as lfp$(\psi(\cdot))$ under the ψ function:*

$$\psi(\pi) = \begin{cases} \pi & \textit{if } \gamma(s_0, \pi) \in \beta^\star \\ \pi.\alpha & \textit{if } \gamma(s_0, \pi) \notin \beta^\star \textit{ for an executable } \alpha \textit{ with } \gamma(s_0, \pi.\alpha) \notin \widehat{\gamma}(s_0, \pi) \\ \textit{failure} & \textit{if } \gamma(s_0, \pi) \notin \beta^\star \textit{ and no such } \alpha \textit{ exists.} \end{cases}$$

Lemma 9. *For any GR problem P, it holds that lfp$(\psi(P)) =$* `Universal`(P)*, where ψ is as in Definition 11 and* `Universal` *is Algorithm 5.*

Fact 10. `Universal` *is sound and complete for* **P***, and has the termination property.*

Despite being complete, the main disadvantage of `Universal` is that it never aims at the current goals —except at solution states. This is problematic for *weakness of will* problems.

Example 10 (Akrasia)**.** *When you are not browsing the internet, you can and want to work. When you are browsing the internet, you desire not to work. Thus, either working or browsing the internet leave you in a state of satisfaction. Now you find yourself neither working nor browsing the internet.* Let us formalize these elements:

$$s_0 = \{-work, -internet\}$$

$$\mathcal{G} = \left\{ \begin{matrix} (-internet, work), \\ (internet, -work) \end{matrix} \right\} \qquad A = \left\{ \begin{matrix} \{\top\}\ \mathsf{browse}\ \{internet\}, \\ \{-internet\}\ \mathsf{work}\ \{work\} \end{matrix} \right\}$$

Let $\beta = \beta_0$ be induced by the normative system $(\vdash_0, \mathcal{G})$ and define the GR problem $P = (S, A, s_0, \beta)$. The two plans for P are the akrasia-friendly solution browse or the more intuitive solution work that fulfils the initial goal. See Figure 4 for an illustration of this. Based on the informal description you seem to keep akrasia under control, while in fact:

- `Universal` (Algorithm 5) returns either browse or work.
- `OnlineGR`, `BDI` and the other `OffGR` planners (Algorithms 4, 6–9) return work.

5.3 `UniClass`*

`UniClass`* aims for an initial goal state (like β`Classical`*) but also adds a test for $\beta^\star$ (like `Universal`). This makes it sound and also to return the best solution in Example 10.

Definition 12 (`UniClass`*)**.** `UniClass`* *is defined as lfp*$(\psi(\cdot))$ *under the* ψ *function:*

$$\psi(\pi) = \begin{cases} \varphi(s_0, \beta(s_0)) & \text{if } \pi = \emptyset \\ \pi & \text{if } \pi \neq \emptyset \text{ and } \gamma(s_0, \pi) \in \beta^\star \\ \text{failure} & \text{otherwise.} \end{cases}$$

Lemma 11. *For any GR problem P, it holds that lfp$(\psi(P)) =$ `UniClass`$^*(P)$, where ψ is as in Definition 12 and* `UniClass`* *is Algorithm 6.*

Corollary 12. `UniClass`$^*(P) \supseteq \beta$`Classical`$^*(P) \cap$ `Universal`(P), *for any $P \in \mathbf{P}$.*

The completeness class of `UniClass`* (see next) is that of β`Classical`* (Prop. 8(ii)).

Theorem 13. `UniClass`* *is sound and has the termination property.* `UniClass`* *is complete for $P = (S, A, s_0, \beta)$ iff whenever $\beta^\star \cap S^* \neq \emptyset$, then also $\beta^\star \cap S^* \cap \|\beta(s_0)\| \neq \emptyset$.*

Corollary 14. `UniClass`* *is complete for all problems $P = (S, A, s_0, \beta)$ satisfying (i)–(ii):*

(i) β is decreasing from s_0 $\qquad R_\beta(s_0, s)$ *implies* $\beta(s_0) \supseteq \beta(s)$

(ii) all initial goal states are reachable $\qquad R_\beta(s_0, \cdot) \subseteq R^*(s_0, \cdot)$.

6. `OffGR` Planners: β-methods

When should an agent reconsider its goal? According to `OnlineGR`, this should be done 'upon any relevant surprise' while for `BDI` 'only after the current intention is fulfilled or abandoned'.[14] In `OffGR` there is 'not much to reconsider upon a solution state' (Eq. 10, Fact 5). Still, a pure search for solution states can fail to achieve obvious goals (Ex. 10).

This tension between solutions and goals has been observed elsewhere in studies of practical rationality.[15] Among `OffGR` planners, this tension manifests itself as a trade-off between completeness class and goal-enforcing power.

14. Thus, `BDI` agents dismiss short-term goals (e.g. to indulge oneself) as they might render one's intention no longer desirable or achievable (e.g. to recover from an addiction). An opposite policy enables `OnlineGR` to address highly uncertain scenarios with some prospects for goal fulfilment.
15. Game theorists have also observed a similar tension in the prisoner's dilemma: the overall best state for the two players (solution) is incompatible with their strategic incentives (goals).

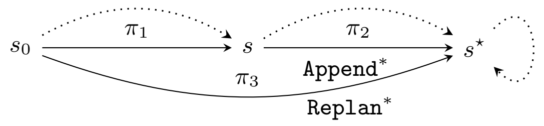


Figure 5: `Append`* and `Replan`* update goals after each successful call to φ. In this figure, both planners find first a plan π_1 to an initial goal state s. `Append`* finds then a plan π_2 from s to $s^\star \in \beta^\star$ and returns $\pi_1.\pi_2$. `Replan`* instead discards π_1, finds a new plan π_3 from s_0 to $s^\star$ and returns π_3. (Note that $\pi_3 = \pi_1.\pi_2$ is also possible).

On the latter, certainly there are advantages to plans that regularly pursue tangible goals, as these plans (1) comply with the norms that trigger such goals, (2) are better motivated and (3) their search can speed up with the use of heuristic functions.

Going back to our initial question, β-methods are `OffGR` planners that reconsider the goal after every plan, but not inside each plan. The two β-methods we study differ on what to do with the current plan. `Append`* keeps this plan and expands it after each goal update. `Replan`*, in contrast, drops the plan after each goal update and starts a new search from the initial state aimed the new goal. See Figure 5 for a comparison.

6.1 `Append`*

Let us consider first an `OffGR` planner that appends plan after plan.

Definition 13 (`Append`*). `Append`* *is defined as lfp($\psi(\cdot)$) under the ψ function:*

$$\psi(\pi) = \pi.\varphi\big(\gamma(s_0, \pi), \beta(\gamma(s_0, \pi))\big).$$

Observe that the initial value for this function ψ is just $\psi(\emptyset) = \varphi(s_0, \beta(s_0))$.[16]

Lemma 15. *For any GR problem P, it holds that lfp($\psi(P)$) =* `Append`*(P)*, where ψ is as in Definition 13 and* `Append`* *is Algorithm 7.*

Observe that `Append`* might get stuck alternating between two (mutually reachable) goal states, and so it might not terminate.

Theorem 16. `Append`* *is sound but does not have the termination property.* `Append`* *is complete for $P = (S, A, s_0, \beta)$ iff $(R_\beta \cap R^*)^*(s_0, \cdot) \cap \beta^\star \neq \emptyset$ whenever a solution for P exists.*

As a consequence of Theorem 16, `Append`* is complete for any problem whose solution states can be achieved while pursuing goals.

Corollary 17. `Append`* *is complete for any GR problem $P \in \mathbf{P}$ satisfying the inclusion $\beta^\star \subseteq (R_\beta \cap R^*)^*(s_0, \cdot)$.*

Despite the negative result in Theorem 16, one can enforce the termination property on `Append`* with a slight modification: that it maintains a list of visited goal states.

16. With detail, $\psi(\emptyset) = \emptyset.\varphi(\gamma(s_0, \emptyset), \beta(\gamma(s_0, \emptyset))) = \varphi(s_0, \beta(s_0))$.

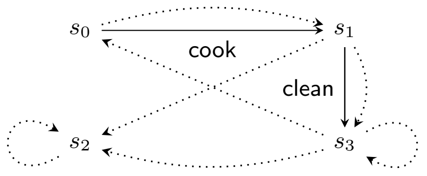


Figure 6: The Möbius Strip from Example 11. The action transitions $\{(s_i, s_1)\}_{1\leq i\leq 3}$ for cook and $\{(s_0, s_0), (s_2, s_2)\}$ for clean are omitted for clarity.

Lemma 18. `Append`**, as defined by Algorithm 7 with %-lines, has the termination property.*

`Append`* performs better than `UniClass`* whenever the progression towards a solution does increase the goals. This is the case under the so-called *Möbius strip*: a chaining of norms such as $(\top, p), (p, q), (q, \neg p)$ with a conflict between its first and last norms.[17]

Example 11 (Möbius Strip)**.** *If you are hungry then you ought to cook (and eat). After cooking, you should clean up. After cleaning up, you ought (temporarily) not to cook. You find yourself hungry, you have not cooked, and the kitchen is clean.* (Makinson, 1999). Let us formalize this scenario as follows:

$$\begin{aligned} s_0 &= \{hungry, clean, -cook\} \\ A &= \left\{\begin{array}{c}\{\top\}\ \mathsf{cook}\ \{-hungry, cook, -clean\},\\ \{-clean\}\ \mathsf{clean}\ \{clean, -cook\}\end{array}\right\} \end{aligned} \qquad \mathcal{G} = \left\{\begin{array}{c}(hungry, cook),\\ (cook, clean),\\ (clean, -cook)\end{array}\right\}$$

Note that clean resets the variable $cook$ to false. Let $P = (S, A, s_0, \beta)$ be the GR problem induced by the above elements and the intuitive goal function β defined by:

$$\begin{array}{llll} s_0 & \xrightarrow{\beta} \{cook\} & s_1 = s_0.\mathsf{cook} & \xrightarrow{\beta} \{clean\} \\ s_2 = \{hungry, cook, clean\} & \xrightarrow{\beta} \{-cook\} & s_3 = s_0.\mathsf{cook}.\mathsf{clean} & \xrightarrow{\beta} \{-cook\}. \end{array}$$

See also Figure 6 for an illustration. The `OffGR` planners give the following outputs:

- $\{\beta$`Classical`*, `UniClass`$^*\}$ return cook resp. *failure* after a call $\varphi(s_0, \beta(s_0)) = \mathsf{cook}$.
- $\{$`Append`*, `Replan`*, `Universal`, `OnlineGR`, `BDI`$\}$ return the solution $\mathsf{cook}.\mathsf{clean}$.
- $\{\beta$`Saturate`$\}$ returns *failure* as it cannot reach the desired (saturated) state s_2.

6.2 `Replan`*

This planner searches for a new plan from scratch after every goal update $s \longmapsto \beta(s)$.

Definition 14 (`Replan`*)**.** `Replan`* *is defined as lfp$(\psi(\cdot))$ under the ψ function:*

$$\psi(\pi) = \varphi\big(s_0, \beta(\gamma(s_0, \pi))\big).$$

17. The chaining in a Möbius strip illustrates the detachment of inconsistent goals under deontic detachment (DD). The logic $\vdash$ based on (FD)+(DD) gives here the inconsistent goal $\{p, q, \neg p\}$.

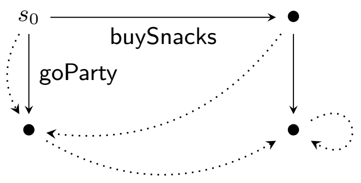


Figure 7: Unwilling Snacks, from Example 12.

Lemma 19. *For any GR problem P, it holds that $lfp(\psi(P)) = \texttt{Replan}^*(P)$, where ψ is as in Definition 14 and $\texttt{Replan}^*$ is Algorithm 8.*

Again, $\texttt{Replan}^*(P)$ might not terminate when cycles exist in the goal relation R_β.

Theorem 20. $\texttt{Replan}^*$ *is sound but does not have the termination property.* $\texttt{Replan}^*$ *is complete for $P = (S, A, s_0, \beta)$ iff whenever a solution for P exists, then $T(s_0, \cdot) \cap \beta^\star \neq \emptyset$, where T is the relation inductively defined by:*

$$T_0 = Id_S \quad \text{and} \quad T_{n+1} = (T_n \mid R_\beta) \cap R^* \quad \text{and} \quad T = \textstyle\bigcup_{n\in\omega} T_n.$$

Let us call a state in $R^*_\beta(s, \cdot)$ a state **remotely desirable** at s. It suffices for $\texttt{Replan}^*$ to be complete that solution states are remotely desirable, and that the latter are reachable.

Corollary 21. $\texttt{Replan}^*$ *is complete for P whenever $\beta^\star \subseteq R^*_\beta(s_0, \cdot) \subseteq R^*(s_0, \cdot)$.*

$\texttt{Replan}^*$ outperforms $\texttt{Append}^*$ when every solution reached by goal pursuits needs to be revised at some point. Consider the next variant of the running example.

Example 12 (Unwilling Snacks)**.** Let us revise the party scenario (Ex. 5) as follows. *You dislike the idea of bringing snacks (even after buying them!). Only at the party you will realize that bringing snacks was a nice gesture.* Let the set of norms now be:

$$\mathcal{G} = \left\{ \begin{array}{c} (\top, \mathit{meet}), \\ (-\mathit{party}, -\mathit{snacks}), \\ (\mathit{party}, \mathit{snacks}) \end{array} \right\}$$

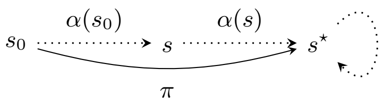
Define $P = (S, A, s_0, \beta)$ as in Examples 4–6 except that $\beta = \beta_0$ is now induced by the set $\mathcal{G}$ just defined. See Figure 7 for an illustration. Note that the initial and solution states are path-connected twice: via R (⭢) and via R_β (↳). Alas, these two paths are disjoint.

Now, both $\{\texttt{Append}^*, \texttt{Replan}^*\}$ consider the plan goParty first. After updating the goal, $\texttt{Append}^*$ cannot expand this plan into a solution while $\texttt{Replan}^*$ does revise it into buySnacks.goParty. Considering also the other algorithms, we have:

- $\{\beta\texttt{Classical}^*, \texttt{UniClass}^*, \texttt{Append}^*\}$ return *failure*.
- $\{\texttt{OnlineGR}, \texttt{BDI}\}$ execute `goParty` and then return *failure*.
- $\{\texttt{Universal}, \texttt{Replan}^*, \beta\texttt{Saturate}\}$ return buySnacks.goParty.

Finally, one can also modify $\texttt{Replan}^*$ as usual so as to enforce the termination property.

Lemma 22. $\texttt{Replan}^*$*, as defined by Algorithm 8 with %-lines, has the termination property.*



$$\begin{tikzcd} s_0 \arrow[r, dotted, "\alpha(s_0)"] \arrow[rr, bend right, "\pi"'] & s \arrow[r, dotted, "\alpha(s)"] & s^\star \arrow[loop right, dotted] \end{tikzcd}$$

Figure 8: β`Saturate` updates s_0 with $\alpha(\cdot)$-actions to produce states that are remotely desired at s_0. If this procedure reaches a solution state $s^\star$, then φ is invoked over the original domain Σ for a plan towards this state $s^\star$.

7. `OffGR` Planners: β-saturation

The third class of `OffGR` planners incorporate a search for β's fixed points before planning. To this end, one defines an inductive method for the the *goal* ⚖ in the naive pipeline (1).

7.1 β`Saturate`

For a GR problem $P = (S, A, s_0, \beta)$, β`Saturate`(P) iteratively updates each state s (initially $s = s_0$) with its goals $\beta(s)$ up to a solution state, at which point an invocation of φ = `Classical` is made. The solution state thus obtains by saturating s_0 with goals (Figure 8).

Technically, this is done by relaxing the original problem P. We expand the planning domain (S, A) into a domain $(S, A \cup A')$ containing new actions $A' = \{\alpha(s) : s \in S\}$. This set A' contains a (nameless) **dummy action** $\alpha(s)$ that produces $\beta(s)$ only on state s:

$$\alpha(s) = \big(pre(\alpha(s)), \mathit{eff}(\alpha(s))\big) = (s, \beta(s)).$$

Observe that if $s \in \beta^\star$, then $\alpha(s) = (s, \beta(s))$ is an idle action, equivalent to $(s, \emptyset)$. Henceforth, we identify any such idle action $\alpha(s)$ with the (sub)plan $\emptyset$.

Definition 15 (β`Saturate`)**.** *β`Saturate` is defined as $lfp(\psi(\cdot))$ under the ψ function:*

$$\psi(\pi) = \begin{cases} \pi.\alpha(\gamma(s_0, \pi)) & \text{if } \gamma(s_0,\pi) \notin \beta^\star \\ \varphi(s_0, \gamma(s_0, \pi)) & \text{if } \gamma(s_0,\pi) \in \beta^\star \text{ and } \pi \notin A^* \\ \pi & \text{if } \gamma(s_0,\pi) \in \beta^\star \text{ and } \pi \in A^* \end{cases}$$

The β**saturation** of s_0, denoted $R^\diamond(s_0)$, is the $\subseteq$-minimal set containing $s_1 = \gamma(s_0, \alpha(s_0))$ and closed under the map $s_n \longmapsto s_{n+1} = \gamma(s_n, \alpha(s_n))$. Observe that $R^\diamond(s_0) \subseteq R^+_\beta(s_0, \cdot)$.

Example 13 (Forget the Snacks!)**.** Let us revise once more the party scenario (Ex. 5). *You are too considerate, and while at home you cannot conceive not bringing snacks to the party —even worse, the shops are closed now. This attitude is unfounded, though, as all party-goers (and you, once at the party) agree that snacks are lame.* Let us expand the language $\mathcal{L}$ from Ex. 5 with a new atom *open* (= *the snack shop is open*) and define:

$$s_0 = \{-open, -party, -snacks, -meet\} \qquad \mathcal{G} = \left\{ \begin{array}{c} (\top, meet), \\ (-party, snacks), \\ (party, -snacks) \end{array} \right\}$$

$$A = \left\{ \begin{array}{l} \{-party\}\ \mathsf{goParty}\ \{party\}, \\ \{open\}\ \mathsf{buySnacks}\ \{snacks\} \end{array} \right\}$$

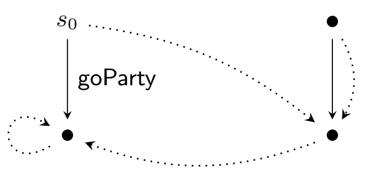


Figure 9: Forget the Snacks! from Example 13.

Let $P = (S, A, s_0, \beta)$ be defined from these elements and the goal function $\beta = \beta_0$. Figure 9 shows how β`Saturate`(P) identifies the solution state by following the R_β-path from s_0. Other planners fail as the solution state is not a goal state of any reachable goal state:

- $\{\beta$`Classical`*, `UniClass`*, `Append`*, `Replan`*, `OnlineGR`, `BDI`$\}$ return *failure*.
- $\{$`Universal`, β`Saturate`$\}$ return the solution `goParty`.

Lemma 23. *For any GR problem P, it holds that lfp$(\psi(P)) = \beta$*`Saturate`*(P), where ψ is as in Definition 15 and β*`Saturate` *is Algorithm 9.*

β`Saturate` is complete for GR problems containing a solution state in $R^\diamond(s_0)$ that is reachable. Observe that at most one solution state exists in any such set $R^\diamond(s_0)$.

Theorem 24. β`Saturate` *is sound but does not terminate.* β`Saturate` *is complete for $P = (S, A, s_0, \beta)$ iff whenever a solution exists, $R^*(s_0, \cdot) \cap \beta^\star \cap R^\diamond(s_0) \neq \emptyset$.*

Once more, one can slightly modify the planner β`Saturate` so as to satisfy termination.

Lemma 25. β`Saturate`*, defined by Algorithm 9 with %-lines, has the termination property.*

7.2 A Variant of β`Saturate`

While β`Saturate` solves the above Example 13, it fails at the following simple example.

Example 14 (Omelette)**.** *Suppose everytime an egg is in your fridge, you desire to cook an omelette. There is exactly one egg in your fridge, and the shops are closed.* Let us define:

$$\begin{aligned} s_0 &= \{egg, -omelette\} \\ \mathcal{G} &= \{(egg, omelette)\} \\ A &= \Big\{ \{egg\}\ \mathsf{cook}\ \{-egg, omelette\} \Big\} \end{aligned}$$

Here, $\beta(s_0) = \{omelette\}$. $\beta^\star = \|\beta(s_0)\| = \{s_1, s_2\}$ where $s_1 = \{egg, omelette\}$ and $s_2 = \{-egg, omelette\}$. Goal saturation $R^\diamond(s_0)$ contains a unique state $s_1 = \gamma(s_0, \alpha(\beta(s_0)))$, but this state is unreachable as you cannot buy an egg now. Thus,

- $\{\beta$`Saturate`$\}$ returns *failure*.
- `OnlineGR`, `BDI` and all other `OffGR` planners return the solution `cook`.

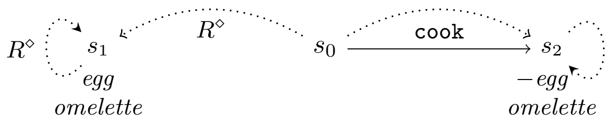


Figure 10: Omelette, from Example 14. The β-saturation steps are labeled with $R^\diamond$.

One can turn β`Saturate` into a less naive planner by removing the remains of the initial state from the β-saturated state $s^\star \in \beta^\star$. This variant is defined by the ψ function:

$$\psi(\pi) = \begin{cases} \pi.\alpha(\gamma(s_0, \pi)) & \text{if } \gamma(s_0, \pi) \notin \beta^\star \text{ and } (\pi = \emptyset \;\text{ or } \pi \notin A^*) \\ \varphi(s_0,\ \gamma(s_0, \pi) \setminus s_0) & \text{if } \gamma(s_0, \pi) \in \beta^\star \text{ and } \pi \notin A^* \\ \pi & \text{if } \gamma(s_0, \pi) \in \beta^\star \text{ and } \pi \in A^* \\ \textit{failure} & \text{if } \gamma(s_0, \pi) \notin \beta^\star \text{ and } \pi \in A^* \setminus \{\emptyset\} \end{cases}$$

Example 15 (Omelette, cont'd)**.** The planner $\mathit{lfp}(\psi(\cdot))$ based on the above ψ function solves Example 14. It turns the saturated state $s_1 = \{egg, omelette\}$ into a goal:

$$\{omelette\} = s_1 \setminus s_0 = \{egg, omelette\} \setminus \{egg, -omelette\}.$$

When this goal is supplied to the local planner, $\varphi(s_0, \{omelette\})$ returns the solution `cook`.

Observe that the classical plan π returned by the invocation $\varphi(s_0,\ s^\star \setminus s_0)$ might not lead to a solution state. This is why we add a $\beta^\star$-membership test in the modified ψ function. This test preserves the soundness of β`Saturate` into this variant.

8. Discussion

Let us start with a recap in Table 1 of the performance of GR algorithms on the examples. This Table illustrates the trade-off between goals ('Norm viol.') and solution states ('No'), particularly among sound `OffGR` planners. With more detail:

- {`Universal`, β`Saturate`} outperform β-methods when pursuing goals is futile (Ex. 13).
- Most `OffGR` planners are incomplete but fulfil goals better than `Universal` (Ex. 10).
- `Append`$^*(P)$ can be expected to fulfil more goals than `Replan`$^*(P)$ when both algorithms are complete for P, but `Replan`* seems to solve more GR problems (Ex. 12).
- β`Saturate` displays an erratic behaviour: from successful (Ex. 13) to naive (Ex. 14).
- `OnlineGR` and `BDI` cannot predict (nor react) to certain norm violations (Ex. 6, 12).

The next sections address pending questions: (8.1) How the completeness classes relate to each other. (8.2) The impact of the local function φ. (8.3) Techniques for combining `OffGR` planners. (8.4) On the reducibility of `OffGR` to classical planning (question Q2). (8.5) The application of GR to deontic logic.

Table 1: A comparison of GR algorithms on examples. The values read as follows: (Yes) The planner returns a solution. (No) The planner returns *failure*. (Unsound) The planner may return a non-solution plan. (Norm viol.) The plan returned might violate an active norm. For all `OffGR` planners, we use their terminating versions.

Ex.	`OnlineGR`, `BDI`	β`Classical`*	`Universal`	`UniClass`*	`Append`*	`Replan`*	β`Saturate`
6	Unsound	Unsound	Yes	Yes	Yes	Yes	Yes
10	Yes	Yes	Norm viol.	Yes	Yes	Yes	Yes
11	Yes	Unsound	Yes	No	Yes	Yes	No
12	Unsound	No	Yes	No	No	Yes	Yes
13	No	No	Yes	No	No	No	Yes
14	Yes	Yes	Yes	Yes	Yes	Yes	No

8.1 A Hierarchy of Completeness Classes

One can first compare `OffGR` planners by their effective range in the class of GR problems.

Proposition 26. *The classes of GR problems for which Algorithms 4–9 are complete can be arranged as follows:*

$$\beta\texttt{Classical}^* = \texttt{UniClass}^* \subset \texttt{Append}^* \subset \texttt{Replan}^* \subset \texttt{Universal}$$

and β`Saturate` $\subset$ `Universal`. β`Saturate` *is incomparable with all other planners.*

Secondly, one can compare the computational cost of generating a solution.

Remark 2. A quick look at the definitions Algorithms 4–9 also shows that these planners demand an increasing number of steps in order to generate a (worst case) solution π for a GR problem $P = (S, A, s_0, \beta)$. These steps consist of computing:

(Algorithm 4)	β`Classical`*	(1) one value of $\beta(\cdot)$ + (2) a classical plan.
(Algorithm 6)	`UniClass`*	(1) + (2) plus (3) a $\beta^\star$-membership test.
(Algorithm 5)	`Universal`	(2) + an $o(\|\pi\|)$ number of steps (3)
(Algorithm 7)	`Append`*	an $o(\|\pi\|)$ number of steps (1)–(3).
(Algorithm 8)	`Replan`*	an $o(\|\pi\|^2)$ number of steps (1)–(3).
(Algorithm 9)	β`Saturate`	an $o(\|S\|)$ number of steps (1) + one step (2).

These complexities partly depend on the parameter $\beta(\cdot)$. For goal functions defined from a normative system $(\mid\!\sim, \mathcal{G})$, computing $\beta(s)$ reduces to the *model checking* problem for the deontic logic $\mid\!\sim$, which is a function of the size of the state s and the set of norms $\mathcal{G}$.

8.2 `Classical` *vs.* `Classical`*

In Section 3, we claimed that `Classical`* brings some advantages to `OffGR` planners, when compared to their `Classical`-based counterparts. Let us define those counterparts first.

Definition 16 (ψ functions based on `Classical`)**.** *For each ψ function from Definitions 10, 12, 13, 14, we define the corresponding ψ function simply by interpreting φ =* `Classical`.

Definition 17 (`OffGR` planners based on `Classical`)**.** *Define the planners* {β`Classical`, `UniClass`, `Append`, `Replan`} *by interpreting φ =* `Classical` *in Algorithms 4, 6, 7 and 8 respectively.*

Local maxima. The first advantage we mentioned in Section 3 was about local maxima. A *local maximum* is: (1) a goal state that is not a solution state (for β`Classical`, `UniClass`) or, more generally, (2) a goal state from which no solution state is remotely desirable (for `Append`, `Replan`) or, finally, (3) a goal state that closes an R_β-cycle back to s_0 (for `Replan`). `Classical`-based `OffGR` planners do get trapped when all solutions traverse a local maximum.

Example 16. Let `Planner` $\in \{$`UniClass`, `Append`, `Replan`$\}$ and consider the GR problem P described by the following search space:[18]

$$\begin{tikzcd}
s_0 \arrow[r, "\alpha_1"] \arrow[r, dotted, bend right] \arrow[rr, dotted, bend right] & s \arrow[r, "\alpha_2"] \arrow[l, dotted, bend right] & s^\star \arrow[loop right, dotted]
\end{tikzcd}$$

State s is a local maximum. Since $\varphi =$ `Classical`, any call $\varphi(s_0, \beta(s_0))$ within an execution of `Planner`(P) will return the plan α_1. The only solution $\alpha_1.\alpha_2$ traverses state s, and so

- `UniClass`(P), in any execution, will return *failure*, since state s is not in $\beta^\star$.
- `Append`(P) also returns *failure*. The first call to φ returns α_1. At state s, the goal state is s_0 but this state cannot be reached from s, so the second call returns *failure*.
- `Replan`(P), in any execution, gets stuck between the plans $\alpha_1 = \varphi(s_0, \beta(s_0))$ and $\emptyset = \varphi(s_0, \beta(s))$. The terminating version of `Replan`(P) necessarily returns *failure*.

In contrast, some executions of $\{$`UniClass`$^*(P)$, `Append`$^*(P)$, `Replan`$^*(P)\}$ return the solution $\alpha_1.\alpha_2$ and the rest return *failure* (for the terminating version of `Replan`*). These `Classical`*-based counterparts thus avoid getting trapped in the local maximum s.

Better completeness classes. A second advantage we mentioned in Section 3 is that `Classical`* provides stronger completeness classes to our `OffGR` planners.

Proposition 27. *For each* `Planner` $\in \{$`UniClass`, `Append`, `Replan`$\}$*, its completeness class satisfies:* `Planner` $\subset$ `Planner`*.

These inclusions are depicted in Figure 11. The completeness class of `Planner`* does not only properly extend that of `Planner`, it also enjoys a more elegant characterization. Without going into too much detail, the completeness class for `Planner` obtains by restricting the class of `Planner`* to those GR problems that contain solution states that are also goal states, of a suitable kind, *and* can be reached without traversing any other goal state. (For `UniClass`, *suitable* = all. For `Append`, *suitable* = remote and reachable.)

18. For details, let (S, A) be generated by the set $\mathsf{At} = \{p, q, r\}$ and the set $A = \{\alpha_1, \alpha_2\}$ with the actions:

$$\{p, \neg q\}\ \alpha_1\ \{\neg p, q, r\} \qquad\qquad \{\neg p, q, r\}\ \alpha_2\ \{p, q\}.$$

The GR problem $P = (S, A, s_0, \beta)$ is then given by $s_0 = \{p, \neg q, \neg r\}$ and the goal function $\beta = \beta_0$ induced by the set of norms $\mathcal{G} = \{(p, q), (p, r), (\neg p, \neg q), (\neg p, \neg r)\}$.

$$\begin{tikzcd}
\texttt{UniClass}^* \arrow[r] & \texttt{Append}^* \arrow[r] & \texttt{Replan}^* \arrow[r] & \texttt{Universal} \arrow[d, leftrightarrow] \\
\texttt{UniClass} \arrow[u] \arrow[r] \arrow[rr, bend left] & \texttt{Append} \arrow[u] \arrow[rr, bend right] & \texttt{Replan} \arrow[u] \arrow[r] & \texttt{Universal}
\end{tikzcd}$$

Figure 11: Completeness classes of `OffGR` planners based on $\texttt{Classical}^*$ (top line) and resp. `Classical` (bottom line). Arrows $\to$ denote proper inclusions $\subset$.

A more intuitive hierarchy. A final advantage is that the completeness classes arrange in a more intuitive way under the interpretation $\varphi = \texttt{Classical}^*$. See again Figure 11 (top line *vs.* bottom line) for a comparison under the two interpretations of φ.

Intuitively, a solution $\pi = \pi_1. \cdots .\pi_n$ returned by appending subplans can be reconstructed by replanning (recall Figure 5). This reconstruction proceeds by a series of plans from scratch that happen to coincide at an initial fragment:

$$(\texttt{Append}^*)\ \pi_1.\pi_2. \cdots .\pi_n \qquad \longrightarrow \qquad (\texttt{Replan}^*)\ \pi_1 \ \mapsto\ \pi_1.\pi_2 \ \mapsto\ \cdots \ \mapsto\ \pi_1.\pi_2. \cdots .\pi_n.$$

This intuition is verified if we interpret $\varphi = \texttt{Classical}^*$, as shown by the proof of $\texttt{Append}^* \subseteq \texttt{Replan}^*$ (Proposition 26), but fails under the interpretation $\varphi = \texttt{Classical}$.

Proposition 28. *The completeness classes of* $\{\texttt{UniClass}, \texttt{Append}, \texttt{Replan}\}$ *arrange as:*

$$\texttt{UniClass} \subset \{\texttt{Append}, \texttt{Replan}\} \subset \texttt{Universal}.$$

The completeness classes of `Append` *and* `Replan` *are incomparable.*

Example 17 ($\texttt{Append} \not\subseteq \texttt{Replan}$)**.** Consider a GR problem P with search space:[19]

$$\begin{tikzcd}
s_0 \arrow[r, "\alpha_1"] \arrow[r, dotted, bend right] \arrow[rr, dotted, bend right=40] & s \arrow[l, dotted, bend right] \arrow[r, "\alpha_2"] \arrow[r, dotted, bend left] & s^\star \arrow[loop right, dotted]
\end{tikzcd}$$

- $\texttt{Append}(P)$: the first call gives $\alpha_1 = \varphi(s_0, \beta(s_0))$ and the second $\alpha_2 = \varphi(s, \beta(s))$; any execution of $\texttt{Append}(P)$ thus returns the solution $\alpha_1.\alpha_2$.
- $\texttt{Replan}(P)$: any execution gets stuck in a replanning cycle between $\alpha_1 = \varphi(s_0, \beta(s_0))$ and $\emptyset = \varphi(s_0, \beta(s))$. (For the terminating version, $\texttt{Replan}(P)$ returns *failure* at this point, as $\emptyset$ is visited twice.)

These arguments on local maxima, completeness classes and the `OffGR` hierarchy support the conclusion that offline GR problems are better addressed by the variant $\texttt{Classical}^*$.

19. With more detail, let (S, A) be generated by the set $\mathsf{At} = \{p, q\}$ and a set $A = \{\alpha_1, \alpha_2\}$ with the actions:

$$\{p, \neg q\}\ \alpha_1\ \{\neg p, q\} \qquad\qquad \{\neg p, q\}\ \alpha_2\ \{p, q\}$$

Define $P = (S, A, s_0, \beta_0)$ by the above (S, A) plus $s_0 = \{p, \neg q\}$ and the set of norms $\mathcal{G} = \{(p, q), (q, p)\}$.

8.3 Unions and Fusions of `OffGR` Planners

One way to mitigate the weaknesses of individual `OffGR` planners is to combine them with the hope of obtaining a better completeness class. An immediate way to combine planners is to use (non-deterministic) choice, denoted ($\texttt{Planner1}{\cup}\texttt{Planner2}$), so that

- an execution of ($\texttt{Planner1}{\cup}\texttt{Planner2}$)($P$) is either an execution of $\texttt{Planner1}(P)$ or an execution of $\texttt{Planner2}(P)$.

From the semantics of choice (Fischer & Ladner, 1979) it follows that for incomparable completeness classes $\texttt{Planner1} \not\subseteq, \not\supseteq \texttt{Planner2}$, this operation improves upon each original completeness class:

$$\texttt{Planner1}, \texttt{Planner2} \ \subset \ \texttt{Planner1}{\cup}\texttt{Planner2}.$$

Consider $\texttt{Append}^* \cup \beta\texttt{Saturate}$ for an illustration. All the examples in Table 1 are solved by this combined `OffGR` planner. But can we do better than choice? Let us consider the piecewise combination of the corresponding ψ functions, that we call *fusion*.

Definition 18 (Fusion). *Let ψ_1, ψ_2 be ψ functions. Their fusion is defined by:*

$$(\psi_1 \otimes \psi_2)(\pi) = (\psi_1(\pi) \cup \psi_2(\pi)).$$

Let us show that $\psi_1{\otimes}\,\psi_2$ is a well-defined ψ function. Rephrase the original ψ functions ψ_1, ψ_2 as a set of pairs *(map, case condition)* $= (\varphi_i(\pi), \theta_i)$ of the form:

$$\psi_1(\pi) = \{(\varphi_i(\pi), \theta_i)\}_{i \le k} \qquad \text{and} \qquad \psi_2(\pi) = \{(\varphi'_j(\pi), \theta'_j)\}_{j \le m}.$$

Then, the fusion $(\psi_1 \otimes \psi_2)(\pi)$ from Definition 18 is explicitly defined by the set of pairs:

$$\big(\psi_1{\otimes}\,\psi_2\big)(\pi) = \big\{\big((\varphi_i \cup \varphi'_j)(\pi),\, \theta_i \wedge \theta'_j\big)\big\}_{i \le k,\, j \le m}$$

Now, the set $\{\theta_i\}_{i \le k}$ induces a partition of the logical space S. That is, it splits S into classes that are: *mutually exclusive* $\|\theta_i\| \cap \|\theta_{i'}\| = \emptyset$ for any $i \neq i'$, and *jointly exhaustive* $S = \bigcup_{i \le k} \|\theta_i\|$. And similarly for the set $\{\theta'_j\}_{j \le m}$. Hence, the same is true for their combination, the set $\{\theta_i \wedge \theta'_j\}_{i \le k,\, j \le m}$. As a consequence, $\psi_1{\otimes}\,\psi_2$ is also a ψ function.

For incomparable completeness classes, say $\texttt{Planner1} \not\subseteq, \not\supseteq \texttt{Planner2}$, fusion does indeed improve on choice:

$$\texttt{Planner1}{\cup}\texttt{Planner2} \ \subset \ \texttt{Planner1}{\otimes}\texttt{Planner2}$$

For the inclusion $\subseteq$, note that an execution of $\texttt{Planner1}(P)$ is obviously an execution of $\texttt{Planner1} \otimes \texttt{Planner2}$ that systematically chooses φ_i over φ'_j whenever the corresponding condition $\theta_i \wedge \theta'_j$ holds. (And similarly for $\texttt{Planner2}(P)$.) That any such inclusion is proper $\subset$ can be shown by constructing a suitable example.

While we leave for future work experimental studies of fusions of `OffGR` planners, one can expect that each planner contributes to a fusion in its own way, such as an increase in the number of goals fulfilled ($\texttt{Append}^*$) or an expansion of the completeness class (`Universal`).

8.4 On the Reducibility of Offline GR to Classical Planning

Let us finally address question (Q2) on whether offline GR reduces to classical planning or some extension thereof. Can one compile a GR problem P into a norm-free planning problem P' and then solve P' with existing planners? Our results will map the class $\mathbf{P}_0$ of all β_0-based GR problems into the class $\mathbf{P}_{\triangleright T}$ of planning problems defined by conditional effects and derived predicates —that is, a fragment of PDDL (Haslum et al., 2019). We denote by $\texttt{Planner}_{\triangleright T}$ a generic planner for this class $\mathbf{P}_{\triangleright T}$.

Definition 19 (Conditional effect)**.** *A conditional effect of an action α is any expression of the form $\varepsilon = \theta \triangleright \ell$, where $\theta \subseteq \mathcal{L}$ and ℓ is a literal. In this case, we write $\varepsilon \in \mathit{eff}(\alpha)$.*

A conditional effect $\theta \triangleright \ell$ of α is interpreted as follows: *if θ was true before executing the action ($s \models \theta$), ℓ becomes true afterwards ($\gamma(s, \alpha) \models \ell$).* Otherwise, ℓ is not affected by α.

Definition 20 (Derived predicate)**.** *A derived predicate is an axiom in implication form $\ell \to \ell'$, read as: ℓ' derives from ℓ. A set of derived predicates, or theory, T is part of the planning domain, e.g. $\Sigma = (S, A, T)$, and restricts the possible states to the set $S = \|T\|$.*

The update function γ is revised to accommodate the conditional effects $\theta \triangleright \ell$ of each α (if $s \models \theta$ then $\gamma(s, \alpha) \models \ell$) and all derived predicates $\ell \to \ell'$ (if $\gamma(s, \alpha) \models \ell$ then $\gamma(s, \alpha) \models \ell'$).

Our compilations of a GR problem $P = (S, A, s_0, \beta_0)$ expand its language $\mathcal{L} \longmapsto \mathcal{L}_{\triangleright T}$ and produce planning problems P' in $\mathbf{P}_{\triangleright T}$ with a fixed goal $g = \{\mathit{final}\}$, and auxiliary actions $A' = \{\texttt{end}, \ldots\}$ expressing different ways to achieve this goal g. A plan π for P' is translated back into a plan for P by removing all the auxiliary actions. Let then:

$$\pi \upharpoonright A \quad = \quad \text{the subsequence of } \pi \text{ that contains only its } A \text{ actions.}$$

Ideally, $\pi \upharpoonright A \in A^*$ will be a solution for P.

Definition 21 (Compilation of GR)**.** *A compilation is a map $(\cdot)' : \mathbf{P} \longrightarrow \mathbf{P}_{\triangleright T}$ satisfying:*

(Soundness) if π is a plan for P', then $\pi \upharpoonright A$ is a solution for P.

A map $(\cdot)' : \mathbf{P}_0 \to \mathbf{P}_{\triangleright T}$ that is only defined for GR problems of the form $P = (S, A, s_0, \beta_0)$ is called a compilation for β_0.

Our compilations for a particular β_0 encode the (or some) set of norms $\mathcal{G}$ that defines this function β_0 (Def. 2, Fact 1) and consist of maps of the form:

$$P = (S, A, s_0, \beta_0) \quad \longmapsto \quad P' = (S', A \cup A', T, s'_0, \{\mathit{final}\})$$

Example 18. Let $P = (S, A, s_0, \beta_0)$ be as in Example 5. First, we add two atoms: $\mathsf{At}' = \mathsf{At} \cup \{\mathit{needSnacks}, \mathit{final}\}$. Now we define $P' = (S', A \cup A', T, s'_0, g)$ by:[20]

$$\begin{array}{rclcrcl} S' & = & \mathcal{P}(\mathsf{At}') \cap \|T\| & & T & = & \{\mathit{party} \to \mathit{needSnacks}\} \\ s'_0 & = & \{\neg p : p \in \mathsf{At}'\} & & g & = & \{\mathit{final}\} \end{array}$$

$$A \cup A' \quad = \quad \left\{ \begin{array}{ccl} \{\neg \mathit{party}\} & \texttt{goParty} & \{\mathit{meet}, \mathit{party}\} \\ \{\neg \mathit{party}\} & \texttt{buySnacks} & \{\mathit{snacks}\} \\ \{\mathit{meet}\} & \texttt{end} & \{(\mathit{needSnacks}, \mathit{snacks}) \triangleright \mathit{final}\} \\ \{\mathit{meet}\} & \texttt{end2} & \{(\neg \mathit{needSnacks}) \triangleright \mathit{final}\} \end{array} \right\}$$

20. This compilation of Example 2 was suggested to us by an anonymous reviewer.

The search for plans in an execution of $\texttt{Planner}_{\triangleright T}(P')$ can be summarized as follows:

($\texttt{goParty.end}$). $\texttt{goParty}$ produces *needSnacks*, but $\texttt{end}$ has no effects since *snacks* is false.

($\texttt{goParty.end2}$). Similarly, but now $\texttt{end2}$ has no effects since $\neg$*needSnacks* is false.

($\texttt{buySnacks.goParty.end}$). The A-actions produce *snacks* and resp. *needSnacks*, so that $\texttt{end}$ activates the effect $(\mathit{needSnacks}, \mathit{snacks}) \triangleright \mathit{final}$, producing the goal *final*.

($\texttt{buySnacks.goParty.end2}$). The goal *final* is not produced since $\neg$*needSnacks* fails.

The map $\pi \longmapsto \pi \restriction A$ thus preserves plans that solve P' into solutions for P.

Let us generalize the transformation in Example 18 into a *compilation for* β_0. First, auxiliary actions in $A' = \{\texttt{end}, \texttt{end2}, \ldots\}$ take as precondition the initial goals $\beta_0(s_0)$ (e.g. *meet*). Secondly, each conditional norm in $\mathcal{G}$, say (*party*, *snacks*), is rephrased into: a derived predicate (*party* $\to$ *needSnacks*) and a set of conditional effects for norm compliance. Henceforth, we introduce the new atoms as *goal*(*snacks*) instead of *needSnacks* in Example 18.

Definition 22 (Compilation A)**.** *Let* $P = (S, A, s_0, \beta_0)$ *be a GR problem defined over* At*, and with* β_0 *induced by a normative system* $(\mathcal{G}, \vdash_0)$*. Define* $P' = (S', A \cup A', T, s'_0, g)$ *by:*

$$\begin{aligned} \mathsf{At}' &= \mathsf{At} \cup \{\mathit{final}\} \cup \{\mathit{goal}(\ell')\}_{(\cdot,\ell') \in \mathcal{G}} & S' &= \|T\| \\ A' &= \{\texttt{end1}, \ldots, \texttt{end}k\} \text{ for } k = 2^{|T|} & s'_0 &= s_0 \cup \{\neg p : p \in \mathsf{At}' \setminus \mathsf{At}\} \\ T &= \{\ell_j \to \mathit{goal}(\ell'_j) : (\ell_j, \ell'_j) \in \mathcal{G} \text{ and } \ell'_j \notin \beta(s_0)\} & g &= \{\mathit{final}\}. \end{aligned}$$

Action $\texttt{end}j$ *is defined by* $\mathit{pre}(\texttt{end}j) = \beta_0(s_0)$ *and* $\mathit{eff}(\texttt{end}j) = \{(\theta_1, \ldots, \theta_{|T|}) \triangleright \mathit{final}\}$ *where*

$$\theta_j = \begin{cases} \text{either:} & \neg \mathit{goal}(\ell_j) \\ \text{or:} & \mathit{goal}(\ell_j), \ell'_j \end{cases} \quad \text{and} \quad \begin{array}{l} \mathit{eff}(\texttt{end1}),\ \ldots,\ \mathit{eff}(\texttt{end}k) \text{ jointly contain} \\ \text{all combinations of values for } \theta_1, \ldots, \theta_{|T|}. \end{array}$$

For the (FD)-based logic $\vdash_0$, solution states coincide with the condition *no norm violations*, expressed by formulas of the form $\neg\ell \vee \ell' = \neg(\ell \wedge \neg\ell')$, one for each norm (ℓ, ℓ').

Lemma 29. *Let* $P = (S, A, s_0, \beta_0)$ *be a GR problem, with* β_0 *induced by* $(\mathcal{G}, \vdash_0)$*. Then,*

$$s \in \beta^\star \quad \textit{iff} \quad s \models \{(\neg\ell \vee \ell')\}_{(\ell,\ell') \in \mathcal{G}}.$$

Remark 3 (Plan minimality)**.** Auxiliary actions $\texttt{end}i$ in A' prevent the map $\pi \longmapsto \pi \restriction A$ from preserving plan minimality from P' to P. That is, $\texttt{Planner}_{\triangleright T}(P')$ can return a minimal plan $\pi_1.\pi_2.\texttt{end}i$ even when a subplan π_1 extends into some $\pi_1.\texttt{end}j$ that already solves P'. Since the solution $\pi_1.\pi_2$ for P is not minimal, it cannot be returned by $\texttt{OffGR}$ planners.

To fix this, we impose a deterministic search order on $\texttt{Planner}_{\triangleright T}$ that lists all A' actions before all A actions. This suffices to preserve the plan minimality of π into $\pi \restriction A$.

Proposition 30 (Compilation A)**.** *For any* $P = (S, A, s_0, \beta_0)$*, the map* $P \longmapsto P'$ *in Definition 22 is a compilation for* β_0 *satisfying, moreover,* $\texttt{Planner}_{\triangleright T}(P') \restriction A \subseteq \texttt{UniClass}^*(P)$*.*

Let us finally consider a variant of Compilation A. Compilation B encodes all norms into the language $\mathcal{L}_{\triangleright T}$, not just those norms that did not detach an initial goal into $\beta(s_0)$.

Definition 23 (Compilation B). *Let $P = (S, A, s_0, \beta_0)$ be a GR problem defined over* At*, and with β_0 induced by $(\mathcal{G}, \vdash_0)$. Define $P' = (S', A \cup A', T, s_0', g)$ as follows:*

$$\begin{array}{rlrl} \mathsf{At}' = & \textit{as in Definition 22} & S' = & \|T\| \\ A' = & \{\texttt{end1}, \ldots, \texttt{end}k\} \textit{ for } k = 2^{|\mathcal{G}|} & s_0' = & s_0 \cup \{\neg p : p \in \mathsf{At}' \setminus \mathsf{At}\} \\ T = & \{\ell_j \to goal(\ell_j') : (\ell_j, \ell_j') \in \mathcal{G}\} & g = & \{\mathit{final}\}. \end{array}$$

Each $\texttt{end}j$ *is defined by $pre(\texttt{end}j) = \top$ and $\mathit{eff}(\texttt{end}j) = \{(\theta_1, \ldots, \theta_{|\mathcal{G}|}) \triangleright \mathit{final}\}$ where again the effects of all* $\texttt{end}j$ *actions jointly contain all combinations of values of $\theta_1, \ldots, \theta_{|\mathcal{G}|}$.*

Proposition 31 (Compilation B). *For any $P = (S, A, s_0, \beta_0)$, the map $P \longmapsto P'$ in Definition 23 is a compilation for β_0 satisfying, moreover,* $\texttt{Planner}_{\triangleright T}(P') \upharpoonright A = \texttt{Universal}(P)$.

In sum, we studied two intuitive compilations and proved that they: (1) correspond to α-methods that have a small completeness class ($\texttt{UniClass}^*$) or are goal oblivious ($\texttt{Universal}$); (2) could only be proved to exist for the (FD) logic $\vdash_0$; and (3) are defined over a language with conditional effects. In this respect:

(1) Even if compilations exist that correspond to $\texttt{Append}^*$ or $\texttt{Replan}^*$, they might well be impractical. Encoding goal states requires a greater number of auxiliary *landmark* atoms and `landmark` actions (akin to *final* resp. `end`) that are also exponential in $|\mathcal{G}|$.

(2) Extending the current compilations to deontic logics beyond $\vdash_0$ seems also unlikely. Non-monotonic deontic logics (see Section 8.5) deal with intricate norm conflicts that cannot be represented by soft goals or in preference-based planners.

(3) Conditional effects can in turn be compiled away into classical planning, but these transformations either require an exponential number of actions or, following Nebel (2000), do not preserve the delete relaxation heuristics —but see also Katz (2019).

Our results support the claim that the most interesting `OffGR` methods do not reduce to pure planning, at least for any significant decrease in complexity —computational or analytical.

8.5 Applications in Deontic Logic

Solving deontic puzzles equipped only with logical tools leads often to intricate systems. Offline GR can relieve these deontic logics from some of their complexity, as shown below for two well-known puzzles. In `OffGR`, solving the Chisholm paradox no longer needs (DD), as (FD) is enough. And for the Order puzzle, the Brewka-Eiter goal is achieved by pursuing (the much simpler) PDL goals. `OffGR` further solves these two puzzles without restricting (FD) to contexts. A **context** is a set $\tau \subseteq s_0$ of handpicked facts, so that only τ-facts trigger factual detachment:

$$\dfrac{\begin{array}{ll} [\tau\text{-fact}] & p \in \tau \\ [\text{norm}] & q \text{ if } p \end{array}}{\begin{array}{ll} [\text{goal}] & q \end{array}} \ (\text{FD})[\tau]$$

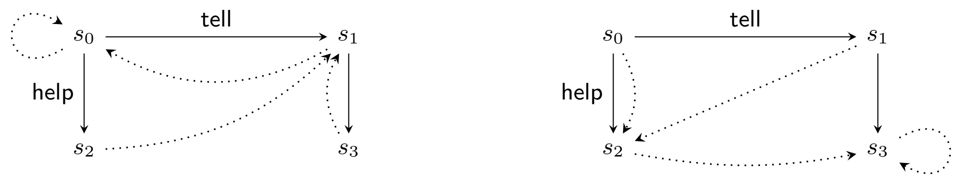


Figure 12: The Chisholm paradox, as a GR problem. (Left) The CTD scenario, where you prefer not to help. (Right) The ATD scenario where you prefer to help.

Chisholm paradox. *You ought to help your neighbours with some task. If you decide to help them, you should call and tell them in advance. If you decide not to help them, you should not call. You decide you are not going to help them.* (Chisholm, 1963).

(*Logic.*) This scenario features a CTD obligation (not to call) under the norms and facts:

$$\mathcal{G} = \{(\top, help), (help, tell), (-help, -tell)\} \qquad s_0 = \{-help, -tell\}.$$

We also consider an alternative scenario where *You decide to help*, featuring an *according-to-duty* (ATD) goal (to call). Since for both scenarios (FD)+(DD) gives an inconsistent goal, this logic is patched into (FD)[τ]+(DD) with a context $\tau \subseteq s_0$ for each scenario:

$\tau = s_0$	CTD : $\tau = \{-help\}$	ATD : $\tau = \emptyset$
$\beta_\omega(s_0) = \{help, tell, -tell\}$	$\beta_\omega(\tau) = \{-tell\}$	$\beta_\omega(\tau) = \{help, tell\}$

(*Offline GR.*) Let us rewind the story and model the preexisting conflict *to help or not* before any decision is made. We add your desire *to not help* and define two actions in A:

$$\mathcal{G}^+ = \mathcal{G} \cup \{(\top, -help)\} \qquad \{-help\}\,\mathsf{help}\,\{help\} \qquad \{-tell, -help\}\,\mathsf{tell}\,\{tell\}.$$

For our deontic logic, define two **preference** relations $\prec \subseteq \mathcal{G}\times\mathcal{G}$, so that the CTD scenario will occur when the norm $(\top, -help)$ is preferred over $(\top, help)$, and viceversa for the ATD scenario. Select the **$\prec$-maximal** elements of $\mathcal{G}^+$ in each case, i.e. the sets $\mathcal{G}_{ctd}, \mathcal{G}_{atd} \subseteq \mathcal{G}^+$:

$$\mathcal{G}_{ctd} = \{(\top, -help), (help, tell), (help, -tell)\} \quad \mathcal{G}_{atd} = \{(\top, help), (help, tell), (help, -tell)\}.$$

Applying (FD) only to the selected norms induces a goal function $\beta = \beta_{ctd}$ or $\beta = \beta_{atd}$ —see Figure 12. Define a GR problem $P = (S, A, s_0, \beta)$ from each β. Then, the planners $\texttt{Replan}^*(P)$ or $\beta\texttt{Saturate}(P)$ return the intuitive solution in each scenario:

$$\text{(CTD): } \pi = \emptyset \qquad \text{(ATD): } \pi = \mathsf{tell.help}.$$

Order puzzle. *Corporal O'Reilly has to follow commands from three superior officers: (1) the Captain orders that in winter the heat should be turned on; (2) the Major orders that in winter the window should be closed; and (3) the Colonel orders that whenever the heat is on the window should be opened.* (Horty, 2007). Let $(k)\colon(\cdot,\cdot)$ denote a norm $(\cdot,\cdot)$ with priority $k \in \mathbb{N}$, where the greater k the stronger the norm. Those commands are then:

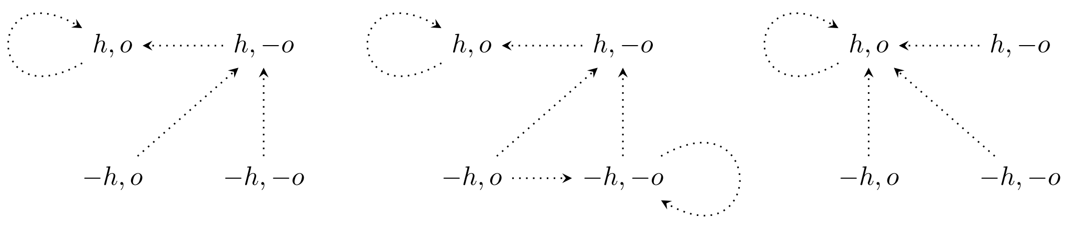


Figure 13: The Order puzzle and the goal relations induced by PDL (left), Hanssen (center) and Brewka-Eiter (right) over states (where $o = open$ and $h = heat$).

$$\mathcal{G} = \{(1)\colon (winter, heat),\ (2)\colon (winter, -open),\ (3)\colon (heat, open)\}.$$

A **prioritized default logic** $\mid\!\sim$, in its deontic variant,[21] selects, for a context $\tau \subseteq \mathsf{Lit}$, a consistent set of norms $\mathcal{G}' \subseteq \mathcal{G}$ based on their strength. The logic $\mid\!\sim$ then detaches all goals from the selected norms $\mathcal{G}'$. (In general, different selections might exist that output rival goal sets, but not under total orders of priorities, like (1) $\prec$ (2) $\prec$ (3) above.)

(*Logic.*) Let us sketch how different logics answer to the puzzle, under context $\tau = \{winter\}$:

PDL (Brewka, 1989). *Iteratively select the strongest applicable and consistent norm, up to an inconsistent goal.* PDL selects norm (2) and then (1). The goal is $\{heat, -open\}$.

Hansen (Hansen, 2008). *Iteratively select the strongest norm, up to an inconsistent goal detachment.* The norms selected are (3) then (2). The goal is $\{-open\}$.

Brewka-Eiter (Brewka & Eiter, 2000). *Guess a goal $X \subseteq G$ such that X is also the PDL output for the pair $(\tau \diamond X, \mathcal{G})$.*[22] The goal is $\{heat, open\}$.

Under each logic, the use of a context τ thus makes the goal state-invariant.

(*Offline GR.*) Without a specified context, (FD)+(DD) applies to all facts of a state s. See Figure 13 for the goal functions $\{\beta_{pdl}, \beta_{ha}, \beta_{be}\}$ induced by the above logics on the search space $S' = \|winter\|$. Notice how the PDL and Hansen goals vary from state to state, but the Brewka-Eiter goal is still state-invariant. For a GR approach, define a set A containing the actions on the window and the heating system:

$$\begin{array}{rcl} \{-open\}\ \mathsf{open}\ \{open\} & \quad & \{-heat\}\ \mathsf{heatOn}\ \{heat\} \\ \{open\}\ \mathsf{close}\ \{-open\} & \quad & \{heat\}\ \mathsf{heatOff}\ \{-heat\}. \end{array}$$

The action transition relation R consists of all horizontal and vertical edges in Figure 13 (not depicted), any two states are mutually reachable. Since no initial state is specified for this puzzle, we let s_0 be any element of $\|winter\|$.

21. For deontic applications, a prioritized default logic differs from its original definition in that the state or context is not part of the output; i.e. the logic $\mid\!\sim$ only concludes goals, not known or inferred facts.

22. The deontic variant of Brewka-Eiter (Liao et al., 2016) drops the requirement that the output X is also a PDL output of $(\tau, \mathcal{G})$. Our definition is a further generalization of it from contexts $(\tau \cup X)$ to states $(\tau \diamond X)$. An update $\tau \diamond X = (\tau \setminus -X) \cup X$ prevents τ-facts from contradicting the guessed goal X.

Fact 32. *Let* $\texttt{Planner} \in \{\texttt{Append}^*, \texttt{Replan}^*, \beta\texttt{Saturate}, \texttt{Universal}\}$ *and* $s \in S'$ *be arbitrary. Define also* $P_{pdl} = (S, A, s, \beta_{pdl})$ *and* $P_{ha} = (S, A, s, \beta_{ha})$*. Then,*

(i) *for any* $\pi \in \texttt{Planner}(P_{pdl})$*, it holds that* $s.\pi \models open \wedge heat$

(ii) *for any* $\pi \in \texttt{Planner}(P_{ha})$*, either* $s.\pi \models open \wedge heat$ *or* $s.\pi \models -open \wedge -heat$

In other words, (i) most `OffGR` planners lead a PDL-guided agent to the Brewka-Eiter goal state. This is a remarkable fact, since the complexity of PDL is Ptime for simple ordered defaults (Kautz & Selman, 1991, Thm. 3), while the complexity of Brewka-Eiter is co-NP-complete (Brewka & Eiter, 1999, Thm. 7.7).

As for (ii), the goal function β_{ha} is the least stable, as it can lead the agent to radically different solution states. (This is even worse for a purely logical approach: a direct plan to the Hansen goal $\{-open\}$ can lead to a *heat* state that violates the strongest norm.)

As a final remark, let us comment on what systems are covered by our notion of **deontic logic** (Section 3). In a nutshell, `OffGR` can recruit any logic for pure deontic reasoning (containing only obligation, permission and constitutive rules). This includes deontic variants of: default logics,[23] defeasible logics, input/output logics, structured argumentation, among many other non-monotonic logics, and perhaps even modal logics (fn. 10). Our goal functions exclude: logics of obligatory actions, and all epistemic or dynamic deontic logics, with observations or preference upgrades. Observations and actions in `OffGR` are extremely simple, as they are taken from classical planning (fn. 6). Classical planning similarly prevents logics with temporal goal specifications: in `OffGR` any temporal constraint between goals or actions (Chisholm paradox) derives from the norms and actions' internal structure. An expansion of `OffGR` with any of these concepts faces the dilemma of modelling the concept as part of the goal selection task (reasoning) or the action selection task (search).

9. Conclusions

We presented the `OffGR` system for offline goal reasoning and argued that it plays an important role, complementary to that of online goal reasoning, for autonomous agency. The need for `OffGR` is especially manifest when goals obtain from reasoning about conflicting hedonistic, moral and legal norms. The use of norms also proved convenient for eliciting and encoding the preferences of an agent. After defining a natural solution concept for `OGRe`, we designed and studied six algorithms for the joint selection of goals and plans. Each algorithm is defined first as the search for solutions of a fixed point equation, and then proved equivalent to an explicit planning program. Some of these search strategies were found to be better built not upon a classical planner but upon a variant of it. We characterized the soundness, completeness and termination of these algorithms, and compared them both theoretically and under a variety of examples. Our preliminary study suggests that plan expansions and replanning might be overall the best strategies for balancing the double aim of intermediate goals and solution states. Finally, we also proved results that strongly suggest the irreducibility of offline GR to planning with conditional effects.

23. Goal functions, by definition, demand a unique goal g to be defined at each state. This condition corresponds to deontic logics that always return a single answer. For rankings of norms, or partial orders, non-monotonic logics can return multiple answers, say $\beta(s) \in \{g_1, \ldots, g_k\}$. In such cases, one can still reason skeptically and define a goal function $\beta_{\cap}(s) = \bigcap \beta(s) = g_1 \cap \cdots \cap g_k$

Offline GR planners cannot, still, foresee all norm violations by themselves, and neither can online GR or BDI algorithms react in time to these violations. Our results thus motivate the search for a switch mechanism between offline and online goal reasoning. This is, we believe, a key problem for the development of agents in our norm-based societies.

Future work. For applications in AI ethics and law, we need to study the impact of `OffGR` planners in deontic logics. As these logics do not study the formation of intentions, it is quite possible that doing so steers current debates on what *detachment principles* apply to law, ethics or desires. For deontic logics that assign priorities or ordinal utilities, moreover, it is an open debate what *aggregation principles* should apply. Thus, experimental studies in GR might decide if the priority of the top goal in a chaining of norms can be defined as the priority of the top norm (last link) or the minimum priority among all norms (weakest link). A related topic is the consideration of *constitutive rules* (Pigozzi & van der Torre, 2018), from which one can detach a legal (or moral) status to particular objects or events; e.g. what legally counts as a vehicle, or as a threat. As such claims are subject to legal debates, this form of reasoning cannot captured by derived predicates in planning.

As a study in GR, the solutions returned by `OffGR` planners can be further improved with *plan repair* techniques that increase the number of goals addressed. At a theoretical level, a study of *goal functions with memory* in offline GR is also an important task. As noticed elsewhere (Cox, 2017; Aha, 2018), diachronic goal inconsistencies might require *goal revision* policies such as a preference for earlier- over later-detached goals. It is noteworthy that deontic logics do not address this problem, as most systems cannot balance goals detached from different timepoints. All these questions on prioritized goals might have in turn an impact on current work in oversubsrciption or preference-based planning.

Another gap in `OffGR` is that solution states do not necessarily exist. What necessarily exists, though, are *solution orbits*, i.e. paths in the goal relation R_β that end up in a cycle. Cyclic plans are indeed part of our daily routines, and in fact we spend considerable thought to optimize these routines, say by minimizing daily norm violations. Adapting `OffGR` planners to return plan orbits is thus also left for future work.

Finally, combining *offline* and *online* search is a problem shared by all disciplines that involve thinking and acting. A formal model for the decision (and wisdom) to switch between these two reasoning modes would mark a milestone for intelligent autonomous agency.

Acknowledgements The authors are thankful to two anonymous reviewers. Their insights helped us correct a proof and improve on the overall content of this article.

Appendix A. Proofs and Algorithms

This Appendix contains explicit definitions of search algorithms (Algorithms 4–9) that correspond each to a ψ-planner $lfp(\psi(\cdot))$ (Definitions 10–15, respectively). We also provide in this appendix proofs of all results from Sections 3–8.

The `OffGR` System.

Fact 1. *Normative systems* $(\mid\!\sim, \mathcal{G})$ *are as expressive as goal functions* $\beta : S \to G$ *when defined over the same set* Lit *of literals.*

Proof. Let NormSys be the class of normative systems $(\mid\!\sim, \mathcal{G})$ with a (single-extension) logic $\mid\!\sim\ \subseteq\ \vdash$ over Lit. Let also GoalFun be the class of (memoryless) goal functions $\beta : S \to G$ with $S \subseteq \mathcal{P}(\mathsf{Lit})$ and $G = \mathcal{P}(\mathsf{Lit})$. Let $t : (\mid\!\sim, \mathcal{G}) \longmapsto \beta$ be the function defined in Equation (6):

$$t((\mid\!\sim, \mathcal{G}))(s) = \beta(s) = \bigcup\{C \subseteq \mathsf{Lit} : (s, \mathcal{G}) \mid\!\sim C\}.$$

(t[NormSys] $\subseteq$ GoalFun.) Immediate from the definition of the induced goal function: $\beta(s) = \{C \subseteq \mathsf{Lit} : (s, \mathcal{G}) \mid\!\sim C\} \subseteq \mathsf{Lit} = G$ for each s. Hence, β is a map of the form $\beta : S \to G$.

(GoalFun $\subseteq t$[NormSys].) Let $\beta : S \to G$ be arbitrary. It suffices to show that there is $(\mid\!\sim, \mathcal{G}) \in$ NormSys such that $t((\mid\!\sim, \mathcal{G})) = \beta$. Consider the (FD) logic $\mid\!\sim\ = \vdash_0$ and the set of norms $\mathcal{G}_\beta = \{(s, \beta(s)) : s \in S\}$. Then:

$$\begin{aligned}
t((\vdash_0, \mathcal{G}_\beta))(s) &= \bigcup\{C \subseteq \mathsf{Lit} : (s, \mathcal{G}_\beta) \vdash_0 C\} && \text{(Def. } t\text{, Eq. 6)}\\
&= \bigcup\{C \subseteq \mathsf{Lit} : s \models B \text{ for some } (B, C) \in \mathcal{G}_\beta\} && \text{(Def. } \vdash_0\text{)}\\
&= \bigcup\{\beta(s) : s \models s\} && ((B, C) = (s, \beta(s)) \text{ in } \mathcal{G}_\beta)\\
&= \bigcup\{\beta(s)\} = \beta(s).
\end{aligned}$$

We just proved that $t((\vdash_0, \mathcal{G}_\beta))(s) = \beta(s)$ holds for arbitrary s, and so we conclude that $t((\vdash_0, \mathcal{G}_\beta)) = \beta$. As a consequence, $\beta \in t$[NormSys] and we are done. $\square$

Proposition 2. *Let $P = (S, A, s_0, g)$ be a classical problem. (Soundness.) If* Classical$^*(P)$ *returns π, then π is a classical plan for P. (Strong completeness.) If $s_0 \not\models g$, then for any non-redundant plan π for P, there is an execution of* Classical$^*(P)$ *that returns π. (Termination.) Any execution of* Classical$^*(P)$ *terminates after a finite number of steps.*

Proof. (Soundness) The soundness of Classical* follows from one of the conditions for Algorithm 1 to exit the **while** loop and return a plan π, namely that $s \models g$ where s takes the value $s = \gamma(s_0, \pi)$.
(Strong completeness.) Let $\pi[1..m] = (\alpha_1, \ldots, \alpha_m)$ be a non-redundant plan for P. Let also the trace of this plan be $\widehat{\gamma}(s_0, \pi) = (s_0, \ldots, s)$ and consider the set of $\|g\|$-states $\{s_1, \ldots, s_n\} = \widehat{\gamma}(s_0, \pi) \cap \|g\|$ listed in increasing order in the trace, i.e. $\widehat{\gamma}(s_0, \pi) = (s_0, \ldots, s_1, \ldots, s_n)$. Let us show that an execution of Classical* exists that returns π. To this end, we prove by induction on k that each subtrace of the form $\widehat{\gamma}(s_0, \pi_k) = (s_0, \ldots, s_k)$ is a run of the **while** loop of an execution of Classical$^*(P)$. (Base case 1.) Since s_1 is the first state in $\widehat{\gamma}(s_0, \pi)$ satisfying $(\cdot) \in \|g\|$, all previous states s in $(s_0, \ldots, s_1)$ satisfy $s \not\models g$ and so they satisfy the disjunctive condition of the **while** loop condition. (Inductive case $k \mapsto k + 1$.) Assume that an execution exists that expands all nodes in $(s_0, \ldots, s_1, \ldots, s_k)$. Since $s_k \models g$, a call is made to Choice. Consider the execution where this call returns true. Hence, the disjunctive condition of the **while** loop is true, and so this execution continues the search towards a $\|g\|$-state by expanding $(s_0, \ldots, s_k, s')$ if an executable action exists at s_k —which we know is true in view of π. Since π is non-redundant, this execution of Classical$^*(P)$ can be extended by expanding the nodes from the above sequence into $(s_0, s_k, \ldots, s_{k+1})$. Observe again that only the last node satisfies the condition $(\cdot) \models g$.

This inductive proof concludes with an execution that expands $(s_0, \ldots, s_n)$. At this point, another call to Choice is made when the current variable $s = s_n$ satisfies $s \models g$.

The execution that returns `false` from this invocation of `Choice` exits the **while** loop and returns π, as desired.
(Termination.) Observe that any execution of $\texttt{Classical}^*(P)$ maintains a list e of expanded nodes and returns failure when a node is visited twice (i.e. when the current node s is already listed in e). This and the fact that the search space is finite jointly imply that an arbitrary execution of $\texttt{Classical}^*(P)$ will terminate in a finite number of steps (and return either *failure* or a plan π). $\square$

Fact 3. *The soundness condition (8) $\gamma(s, \varphi(s, \beta(s))) \models \beta(s)$ holds for classical planning if we interpret $\varphi = \texttt{Classical}$ or $\varphi = \texttt{Classical}^*$.*

Proof. Observe that Condition (8) reduces to (7) $\gamma(s, \pi) \models g$ after we interpret β and φ classically: β as the constant function $\beta_g(\cdot) = g$ (see Definition 3) and $\pi = \varphi(s, g)$ for either $\varphi = \texttt{Classical}$ or $\varphi = \texttt{Classical}^*$. Thus, in order to prove (8) it suffices to prove (7).
($\varphi = \texttt{Classical}$.) Consider the classical problem $P = (S, A, s, g)$. Then, (7) simply follows from the soundness of the classical forward planner $\texttt{Classical}(P) = \varphi(s, g)$ for, in particular, the classical problem P.
($\varphi = \texttt{Classical}^*$.) Now we interpret $\beta = \beta_g$ as before and $\varphi = \texttt{Classical}^*$ as in (9). (Case $s_0 \models g$.) The output of $\varphi(s_0, g)$ is now $\pi = \emptyset$, which clearly satisfies (7) for $s = s_0$ and hence also (8). (Case $s_0 \not\models g$.) $\varphi(s, g)$ returns a value π when `Choice` returns `false` and $s = \gamma(s_0, \pi) \models g$. Thus, the condition $s \models g$ that exits the **while** loop in Algorithm 1 amounts to $\gamma(s_0, \pi) \models g$ which is just Equation (7). Thus, we conclude that Equation (8) holds for the classical planner $\texttt{Classical}^*$ under the reading $\beta(\cdot) = \beta_g(\cdot) = g$. $\square$

`OGRe` Planners: General View.

Fact 4 (Classical to GR)**.** *Each classical problem (S, A, s_0, g) reduces to the GR problem (S, A, s_0, β_g) defined by the constant goal function $\beta_g(s) = g$, for any $s \in S$.*

Proof. One can reason as follows (we omit the condition $\pi \in A^*$ at every step):

$$\begin{array}{llll}\pi \text{ is a solution for } (S, A, s_0, \beta_g(s_0)) & \Leftrightarrow & s_0.\pi \models \beta_g(s_0.\pi) & \text{(Def. solution)} \\ & \Leftrightarrow & s_0.\pi \models g & (\beta_g(\cdot) = g) \\ & \Leftrightarrow & \pi \text{ is a plan for } (S, A, s_0, g) & \text{(Def. plan for } P).\end{array}$$

$\square$

Fact 5. *Let $\varphi = \texttt{Classical}$ or $\varphi = \texttt{Classical}^*$. Then, $s \in \beta^\star$ iff $\varphi(s, \beta(s)) = \emptyset$.*

Proof. Recall that $s \in \beta^\star$ iff $s \models \beta(s)$. Consider the classical problem $P = (\Sigma, s, \beta(s))$ with goal $g = \beta(s)$.
($\varphi = \texttt{Classical}$.) ($\Rightarrow$) This follows from the exit condition $s \models \beta(s)$ for the `while` loop in $\texttt{Classical}(\Sigma, s, \beta(s)) = \varphi(s, \beta(s))$. ($\Leftarrow$) By the soundness of the forward planner `Classical` over classical planning problems (see e.g. (Ghallab et al., 2004) for a proof).
($\varphi = \texttt{Classical}^*$.) ($\Rightarrow$) If $s \models \beta(s) = g$, then $\texttt{Classical}^*$ immediately returns $\pi = \emptyset$ and so $\varphi(s, \beta(s)) = \emptyset$. ($\Leftarrow$) By the soundness property in Proposition 2, we obtain $s = \gamma(s, \emptyset) \models \beta(s)$ and so $s \in \beta^\star$. $\square$

Algorithm 4 β`Classical`*

```
Input: $(S, A, s_0, \beta)$
Output: a plan $\pi$ or failure
1: set $g \leftarrow \beta(s_0)$
2: $\pi \leftarrow$ Classical$^*(S, A, s_0, g)$
3: return $\pi$
```

Fact 6. *If an* `OffGR` *planner* `Planner` *is sound, complete and satisfies termination (Definitions 6–8), then a determinstic execution of* `Planner`(P) *returns a solution for P iff a solution for P exists.*

Proof. ($\Rightarrow$) This is immediate: the solution returned is a witness that a solution exists. ($\Leftarrow$) Suppose a solution for P exists. By completeness, an execution of `Planner`(P) exists that returns a solution π. By termination, all possible runs of `Planner`(P) terminate after finitely-many steps, and a quick look at any `OffGR` planner shows that it returns either a plan or *failure*. Consider an exhaustive execution of runs of `Planner`(P) —each run chosen based on the deterministic search method and an ordering of A. If a run of `Planner`(P) returns a plan, then by soundness this plan is a solution and so `Planner`(P) returns a solution. If a run of `Planner`(P) returns *failure*, the algorithm proceeds to the next try. Since the number of possible runs is countable, eventually the algorithm will terminate and return a solution for P. $\square$

β`Classical`*.

Lemma 7. *Let ψ be the function from Definition 10 and let β*`Classical`* *be the planner from Algorithm 4. For any GR problem $P = (\Sigma, s_0, \beta)$,*

$$\mathit{lfp}(\psi(P)) = \texttt{Classical}^*(P') = \beta\texttt{Classical}^*(P)$$

where $P' = (\Sigma, s_0, \beta(s_0))$ is the transformation of P into a classical problem.

Proof. Let us observe first that $\psi(\emptyset)$ is a fixed point; that is, $\psi(\psi(\emptyset)) = \psi(\emptyset)$, which is immediately clear from Definition 10 (case $\pi \neq \emptyset$). From this, we obtain that $\mathit{lfp}(\psi(P))$ must consist of one step $\psi(\pi) = \pi = \psi(\emptyset)$, thereby returning $\psi(\emptyset) = \varphi(s_0, \beta(s_0))$. Given the definition of P' and that $\varphi = \texttt{Classical}^*$, we just showed that $\mathit{lfp}(\psi(P)) = \texttt{Classical}^*(P')$. The remaining identity can be verified by comparing the two algorithms `Classical`$^*(P')$ and β`Classical`$^*(P)$. Indeed, β`Classical`$^*(P)$ reduces to `Classical`$^*(P')$ when one replacing $\beta \in P$ by the goal $g = \beta(s_0) \in P'$ in the input of Algorithm 4. $\square$

Proposition 8 (β`Classical`*). β`Classical`* *has the termination property but is neither sound nor complete for* $\mathbf{P}$. *In fact,*

(i) β`Classical`* *is sound for P iff $P \in \mathbf{P}' = \{(S, A, s_0, \beta) : \|\beta(s_0)\| \cap R^*(s_0, \cdot) \subseteq \beta^\star\}$.*

(ii) β`Classical`* *is complete for P iff if a solution for P exists then a solution to some state in $\|\beta(s_0)\|$ also exists.*

As a consequence, (iii) classical planning is subsumed by β`Classical`**.*

Algorithm 5 `Universal` % a search for the $\beta^\star$ region

```
Input: $(S, A, s_0, \beta)$
Output: a plan $\pi$ or failure
 1: Let $s = s_0$ and $\pi = \emptyset$ and $e = \{s_0\}$        % $e$ is a list of nodes expanded by $\pi$
 2: while $s \not\models \beta(s)$ do
 3:     $A' = \{\alpha \in A : s \models pre(\alpha)$ and $\gamma(s, \alpha) \notin e\}$
 4:     if $A' = \emptyset$ then
 5:         return failure
 6:     else
 7:         choose $\alpha \in A'$
 8:         set $s \leftarrow \gamma(s, \alpha)$ and $\pi \leftarrow \pi.\alpha$ and $e \leftarrow e \cup \{s\}$
 9:     end if
10: end while
11: return $\pi$
```

Proof. For termination, $\psi(\emptyset) = \varphi(s_0, \beta(s_0))$ is always a fixed point. From this and the termination property of `Classical`* (Proposition 2), the termination of β`Classical`* follows immediately. The failure of soundness and completeness is shown by Example 9. β`Classical`* may return `goParty` which is not a solution; in addition, a solution exists.

For (i). We use the abbreviation $S^* = R^*(s_0, \cdot)$. ($\Rightarrow$) Suppose β`Classical`* is sound for P, and let $s \in \|\beta(s_0)\| \cap S^*$ be arbitrary. From $s \in S^*$, there is a plan π with $s_0.\pi = s$. Since β`Classical`* makes a call to `Classical`*, we can ignore all $\|\beta(s_0)\|$-states before s (assume Algorithm 2 always chooses `true` for them). Since $s \in \|\beta(s_0)\|$, Algorithm 4's terminating condition holds, so β`Classical`$^*(P)$ may return π. But then by soundness, $s \in \beta^\star$, as desired. ($\Leftarrow$.) Suppose that $\|\beta(s_0)\| \cap S^* \subseteq \beta^\star$ for a given $P = (S, A, s_0, \beta)$. Let then $\pi \in A^*$ be an arbitrary output returned by β`Classical`$^*(P)$ and also let $s = \gamma(s_0, \pi)$. The terminating condition of Algorithm 4 implies that the node s satisfies $s \models \beta(s_0)$ and, also clearly, $s = s_0.\pi \in S^*$. By these two facts and the assumption $P \in \mathbf{P'}$, we have that $s \in \beta^\star$. Since π was an arbitrary output, we conclude that β`Classical`* is sound for P.

For (ii). ($\Rightarrow$) Suppose that β`Classical`* is complete for P, and moreover that a solution for P exists. Hence, β`Classical`$^*(P)$ returns a solution π; by the terminating condition we also have that $s_0.\pi \in \|\beta(s_0)\|$. Hence, a solution π leading to a $\|\beta(s_0)\|$-state also exists. ($\Leftarrow$.) Let P be an GR problem and assume that a solution for P exists. Hence, a solution π also exists that leads to $\|\beta(s_0)\|$. This implies that a classical plan π exists for the transformation $P' = (S, A, s_0, \beta(s_0))$ of P. By Proposition 2, an execution of `Classical`$^*(P')$ returns π. Finally, by Lemma 7, an execution of β`Classical`$^*(P)$ also returns π, which we know is a solution for P.

(iii) Consider the transformation map from classical to GR problems $(S, A, s_0, g) \longmapsto (S, A, s_0, \beta_g)$, where $\beta_g(\cdot) = g$ is a constant goal function. Observe that any such constant function $\beta_g(\cdot) = g$ verifies the condition $\|\beta(s_0)\| = \beta^\star$. The latter trivially implies two conditions: $\|\beta(s_0)\| \cap R^*(s_0, \cdot) \subseteq \beta^\star$ (soundness); and also that $R^*(s_0, \cdot) \cap \beta^\star \neq \emptyset$ implies $R^*(s_0, \cdot) \cap \beta^\star \cap \|\beta(s_0)\| \neq \emptyset$ (completeness). Hence, for any classical problem (S, A, s_0, g) for which `Classical` is sound and complete (that is, the class of all classical problems), β`Classical`* is sound and complete for the corresponding GR problem (S, A, s_0, β_g). □

`Universal.`

Lemma 9. *For any GR problem P, it holds that lfp$(\psi(P)) =$ `Universal`(P), where ψ is as in Definition 11 and* `Universal` *is as in Algorithm 5.*

Proof. Let $P = (S, A, s_0, \beta)$ be a GR problem. We prove by induction on trace length that the set of nodes that can be expanded by $\psi(\pi)$ is identical to the set of nodes that can be expanded by `Universal`(P) upon $s = s_0.\pi$. With detail, we show that this set is generated by the same conditions, such as executability and novelty of the new node. (Base case.) For length 0, the only plan is $\pi = \emptyset$ and both methods coincide on the unique node s_0. (Inductive case.) Assume the claim holds for π. The proof is by cases on $\psi(\pi)$ in Def. 11:

(Case: $\psi(\pi) = \pi$.) In Definition 11, this case implies $\gamma(s_0, \pi) \in \beta^\star$. This corresponds to the terminating condition $s \models \beta(s)$ for the current node $s = \gamma(s_0, \pi)$ in Algorithm 5. These methods then reach the fixed point $\pi = \psi(\pi)$ and resp. return the value π. In the former, no further aplication $\psi(\pi)$ generates new states; in the latter, the execution terminates.

(Case: $\psi(\pi) = \pi.\alpha$.) The conditions here are: $\gamma(s_0, \pi) \notin \beta^\star$ and α *is executable and leads to a new state*. The first condition implies $A' \neq \emptyset$ which corresponds to the **else** case (line 6). The second condition $\gamma(s_0, \pi.\alpha) \notin \widehat{\gamma}(s_0, \pi)$ is equivalent to $\gamma(s_0, \pi.\alpha) \notin e$ in Algorithm 5, since the list e keeps track of visited nodes (lines 1,8). The two conditions upon α are jointly equivalent to node α being in A' ($\alpha \in A'$, line 3). Thus, the nodes that can be expanded in this case are the same as in the **else** case (line 7).

(Case $\gamma(s_0, \pi) \notin \beta^\star$ and *no executable α leads to a new state*.) A quick look to both methods shows that they return *failure*.

This concludes the inductive proof. From this we obtain that for each possible output π, there is a trace generating π in Algorithm 5 iff π is the value returned by an execution of *lfp*$(\psi(P))$, with ψ as in Definition 11. Thus, *lfp*$(\psi(P)) =$ `Universal`(P). □

Fact 10. `Universal` *is sound and complete for* **P***, and has the termination property.*

Proof. (Soundness) Suppose `Universal`(P) returns a plan $\pi \in A^*$. Let $s = \gamma(s_0, \pi)$. In Algorithm 3, the terminating condition for $\pi \neq$ *failure* is: $s \models \beta(s)$. From this, it follows that $\gamma(s_0, \pi) \in \beta^\star$. (Completeness) This follows from the search space S' being always finite for any `OffGR` problem P. (Termination) This property follows from the finite character of S' and the fact that an execution of `Universal`(P) will never expand the same node twice. □

`UniClass`*.

Lemma 11. *For any GR problem P, it holds that lfp$(\psi(P)) =$ `UniClass`$^*(P)$, where ψ is as in Definition 12 and* `UniClass`* *is Algorithm 6.*

Proof. Let $\pi \in A^*$ be an arbitrary plan. We prove first that $\psi(\pi)$ is identical to a run of the **while** loop in `UniClass`$^*(P)$.
(Case $\pi = \emptyset$.) The first case of the ψ function for `UniClass`* (Definition 12) is identical to that for β`Classical`* (Definition 10), namely $\psi(\emptyset) = \varphi(s_0, \beta(s_0))$. Using the proof of Lemma 7, we obtain that $\gamma(s_0, \pi) \models \beta(s_0)$, or *failure* if no classical plan exists. In any case, the value of π becomes $\varphi(s_0, \beta(s_0))$ for both Algorithm 6 and Definition 12.
(Case $\pi \neq \emptyset$.) The remaining cases for ψ in Definition 12 split further into two.
(Subcase $\gamma(s_0, \pi) \in \beta^\star$.) This is the condition for `UniClass`* to return π. Together with $\pi \neq \emptyset$, it is also the condition for ψ to reach a least fixed point $\psi(\pi) = \pi$.

Algorithm 6 `UniClass`*

```
Input: $(S, A, s_0, \beta)$
Output: a plan $\pi$ or failure
1: set $g \leftarrow \beta(s_0)$
2: $\pi \leftarrow$ Classical$^*(S, A, s_0, g)$
3: if $s \not\models \beta(s)$ then
4:     return failure
5: else
6:     return $\pi$
7: end if
```

(Subcase $\gamma(s_0, \pi) \notin \beta^\star$.) This equivalent to the condition for the **if** case (line 5) to return *failure*. Together with $\pi \neq \emptyset$, it is also the condition for Definition 12 to return *failure*. $\square$

Corollary 12. `UniClass`$^*(P) \supseteq \beta$`Classical`$^*(P) \cap$ `Universal`(P)*, for any* $P \in \mathbf{P}$.

Proof. A quick inspection of Algorithm 6 reveals that `UniClass`* returns a plan π under two conditions: (i) $s = \gamma(s_0, \pi) \models \beta(s_0)$ (lines 1–2); and (ii) $s \models \beta(s)$ (**else** case, line 5). But these are precisely the terminating conditions for Algorithm 4 and resp. Alg. 5 to return π. All other cases return *failure*.

The other direction `UniClass`$^*(P) \subseteq \beta$`Classical`$^*(P) \cap$ `Universal`(P) need not hold: a plan $\pi \in$ `UniClass`$^*(P)$ will satisfy $\pi \in \beta$`Classical`$^*(P)$ and $\gamma(s_0, \pi) \in \beta^\star$, but such π need not be minimal with the latter property, in which case $\pi \notin$ `Universal`(P). $\square$

Theorem 13. `UniClass`* *is sound and has the termination property.* `UniClass`* *is complete for* $P = (S, A, s_0, \beta)$ *iff whenever* $\beta^\star \cap S^* \neq \emptyset$*, then also* $\beta^\star \cap S^* \cap \|\beta(s_0)\| \neq \emptyset$.

Proof. (Termination) This follows from the termination property of β`Classical`* in Proposition 8(i), and the decidability of testing the condition $s \models \beta(s)$.
(Soundness) Let $\pi \in A^*$ be returned by `UniClass`$^*(P)$. Algorithm 6's **else** case (line 5) implies that state $s = s_0.\pi$ is a solution state.
(Completeness) ($\Rightarrow$) Let `UniClass`* be complete for $P = (S, A, s_0, \beta)$ and assume that a solution exists, as otherwise we are done. That is, $\beta^\star \cap S^* \neq \emptyset$. We show that $\beta^\star \cap S^* \cap \|\beta(s_0)\| \neq \emptyset$. By the completeness for P, `UniClass`$^*(P)$ returns a solution. Let then π be an arbitrary solution returned by `UniClass`$^*(P)$. (1) By the soundness of `Classical`*, the plan π returned by the invocation of `Classical`$^*(S, A, s_0, \beta(s_0))$ must satisfy $\gamma(s_0, \pi) \in \|\beta(s_0)\|$. Let then $s^\star = \gamma(s_0, \pi)$. (2) Algorithm 6 returns π only if $s^\star \models \beta(s^\star)$. From (1)–(2) we obtain that $s^\star \models \beta(s_0) \cup \beta(s^\star)$, and together with $s^* = \gamma(s_0, \pi)$, we conclude that $s^* \in \beta^\star \cap S^* \cap \|\beta(s_0)\|$, so this set is not empty.
($\Leftarrow$) Suppose that the righthand condition $\beta^\star \cap S^* \cap \|\beta(s_0)\| \neq \emptyset$ holds for a GR problem P. This clearly implies that a solution for P exists, and so we must prove that an execution of `UniClass`$^*(P)$ returns a solution. Let $s^\star$ be an element in the above set, and since $s^\star \in S^*$ let π be such that $s^\star = \gamma(s_0, \pi)$. Without loss of generality, assume that no other solution state exists in $\widehat{\gamma}(s_0, \pi)$. From $s^\star \in \|\beta(s_0)\| \cap S^*$ and the strong completeness of `Classical`* (Proposition 2), an execution of `Classical`$^*(S, A, s_0, \beta(s_0))$ returns π. From $s^\star \in \beta^\star$, this node $s^\star$ passes the test $s^\star \models \beta(s^\star)$, and so an execution of Algorithm 6 returns π. $\square$

Algorithm 7 `Append`* (add %-lines for termination)

```
Input: $(S, A, s_0, \beta)$
Output: a plan $\pi$ or failure
1: Let $s = s_0$ and $\pi = \emptyset$        % and $e = \{s_0\}$.
2: while $s \not\models \beta(s)$ and $\pi \neq$ failure do
3:     $\pi \leftarrow \pi.\varphi(s, \beta(s))$
4:     $s \leftarrow \gamma(s, \pi)$
5:     % Test_expanded($s$)
6: end while
7: return $\pi$
```

Algorithm 7b `Test_expanded`(s)

```
Input : a node $s$
Output : failure (or nothing)
1: if $s \in e$ then
2:     return failure
3: else $e \leftarrow e \cup \{s\}$
4: end if
```

Corollary 14. `UniClass`* *is complete for the class of problems $P = (S, A, s_0, \beta)$ satisfying (i) and (ii):*

(i) β is decreasing from s_0: $R_\beta(s_0, s)$ *implies* $\beta(s_0) \supseteq \beta(s)$

(ii) s_0's goal states are reachable: $R_\beta(s_0, \cdot) \subseteq R^*(s_0, \cdot)$.

Proof. We show first that (i) implies (i') $R_\beta(s_0, \cdot) \subseteq \beta^\star$. Let $s \in R_\beta(s_0, \cdot)$ be arbitrary. Since β is decreasing from s_0, $\beta(s) \subseteq \beta(s_0)$ and by the definition of $R_\beta(s_0, s)$, $s \models \beta(s_0)$; by the previous inclusion, $s \models \beta(s)$; that is, $s \in \beta^\star$. This shows (i').

Next we show completeness from (i') and (ii), so assume that a solution for P exists. First, note that `UniClass`$^*(P)$ does not return *failure*: let $s \in R_\beta(s_0, \cdot)$. By (ii) $s \in R^*(s_0, \cdot)$ and so a plan π exists to s. By strong completeness, an execution of `Classical`$^*(S, A, s_0, \beta(s_0))$ returns π, and by $s \in \beta^\star$, an execution of `UniClass`$^*(P)$ returns π. Hence `UniClass`* is complete for P. □

`Append`*.

Remark 4. For the `Append`* planner, expressions of the form π.*failure* are henceforth identified with *failure*. Similarly we stipulate that $\gamma(s, \mathit{failure}) = s$.

These two conventions apply to Algorithm 7 to grant that `Append`* exits the **while** loop anytime an invocation of φ returns *failure*. These conventions also apply to the function ψ from Definition 13 for the management of all values *failure*.

Lemma 15. *For any GR problem P, it holds that lfp$(\psi(P)) =$* `Append`$^*(P)$*, where ψ is as in Definition 13 and* `Append`* *is Algorithm 7.*

Proof. We prove a correspondence between the possible values (plans) of ψ and the (goal) states that can be expanded by $\texttt{Append}^*(P)$ for a given GR problem P. The proof is by induction on the number n of applications of ψ. Let $\psi^1 = \psi$ and $\psi^{n+1} = \psi \circ \psi^n$. (Base case ψ^1.) The initial value is the empty plan $\emptyset$. We omit this proof as it is similar to the proof of the inductive case (with $s = s_0$, first and last cases only). (Inductive Case: $\psi^n \mapsto \psi^{n+1}$.) Suppose as inductive hypothesis that the possible values π of $\psi^n(\emptyset)$ coincide with the possible plans generated by $\texttt{Append}^*(P)$ at the n-th iteration of the *while* loop in Algorithm 7. Let $\pi = \psi^n(\emptyset)$ be an arbitrary value and let also $s = \gamma(s_0, \pi)$.

(Case $s \models \beta(s)$.) This case exits the **while** loop in Algorithm 7, and $\texttt{Append}^*(P)$ then returns π, so no new plans are generated at this point. On the other hand, since $\varphi = \texttt{Classical}^*$ we can apply Fact 5 and obtain: $\emptyset = \varphi(s, \beta(s)) = \varphi(\gamma(s_0, \pi), \beta(\gamma(s_0, \pi)))$. Using Definition 13 we reach a least fixed point: $\psi(\pi) = \pi.\varphi(s, \beta(s)) = \pi.\emptyset = \pi$. Again, no new values are generated by $\psi^{n+1}(\emptyset)$.

(Case $\pi = \mathit{failure}$.) That is, $\pi = \pi'.\mathit{failure}$. This condition also exits the *while* loop in $\texttt{Append}^*$ to return $\pi = \mathit{failure}$. Since line 3 of Algorithm 7 is equivalent to the step $\psi^n(\emptyset) \longmapsto \psi^{n+1}(\emptyset)$, we also have for $s' = \gamma(s_0, \pi')$ that $\psi^{n+1}(\emptyset) = \psi(\psi^n(\emptyset))) = \pi'.\varphi(s', \beta(s')) = \pi'.\mathit{failure} = \mathit{failure}$.

(Case $s \not\models \beta(s)$ and $\pi \neq \mathit{failure}$.) Both $\psi^{n+1}(\emptyset)$ and $\texttt{Append}^*(P)$ (line 3 of Algorithm 7) reduce in this case to a single application of ψ in $\psi^n(\emptyset) \longmapsto \psi^{n+1}(\emptyset)$. The possible values that the expression $\pi.\varphi(s, \beta(s))$ can take in Algorithm 7 are exactly the same, namely $\psi(\pi) = \pi.\varphi(\gamma(s_0, \pi), \beta(\gamma(s_0, \pi))) = \pi.\varphi(s, \beta(s))$.

This concludes the inductive case. In all cases, the values of $\psi^{n+1}(\emptyset) = \psi(\pi)$ coincide with those for π in the $(n+1)$-th iteration of **while** in Algorithm 7. Thus, we just showed that $\mathit{lfp}(\psi(P)) = \texttt{Append}^*(P)$. □

Theorem 16. $\texttt{Append}^*$ *is sound but does not have the termination property.* $\texttt{Append}^*$ *is complete for* $P = (\Sigma, s_0, \beta)$ *iff* $(R_\beta \cap R^*)^*(s_0, \cdot) \cap \beta^\star \neq \emptyset$ *whenever a solution for* P *exists.*

Proof. (Lack of termination) Let P be the GR problem defined as follows. $S = \{s_0, s_1\}$ is a state space such that s_0, s_1 are: (i) mutually reachable, say with the actions $s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_0$, and (ii) they are goal states of each other: $\|\beta(s_0)\| = \{s_1\}$ and $\|\beta(s_1)\| = \{s_0\}$. All executions of $\texttt{Append}^*(P)$ have the same form: each iteration of the **while** loop alternates between adding α_0 and adding α_1 to the current plan. Hence, $\texttt{Append}^*(P)$ keeps producing all the initial (finite) fragments of the infinite sequence $(\alpha_0.\alpha_1)^* = (\alpha_0, \alpha_1, \alpha_0, \alpha_1, \ldots)$.

(Soundness) The soundness of Algorithm 7 is granted by the condition $s \models \beta(s)$ for exiting the *while* loop, which is also the terminating condition.

(Completeness.) ($\Leftarrow$) Suppose a solution exists and so let $s^\star \in R^*(s_0, \cdot) \cap \beta^\star$. The assumption implies there is a sequence of states $(s_0, s_1, \ldots, s_n)$ such that for each pair (s_k, s_{k+1}),

$$(1)\ (s_k, s_{k+1}) \in R^* \qquad (2)\ (s_k, s_{k+1}) \in R_\beta \quad \text{and also} \quad (3)\ s_n = s^\star \in \beta^\star.$$

Without loss of generality, suppose s_n is the only state in the sequence satisfying (3). (Case $n = 0$.) This simply gives a sequence (s_0) whose unique element satisfies (3) $s_0 \in \beta^\star$. $\texttt{Append}^*(P)$ here returns the solution $\emptyset$. (Case $n \neq 0$.) Since $s_0 \notin \beta^\star$ any trace of $\texttt{Append}^*(\Sigma, s_0, \beta)$ calls $\varphi(s_0, \beta(s_0)) = \texttt{Classical}^*(\Sigma, s_0, \beta(s_0))$. Let us prove by induction that an execution of $\texttt{Append}^*$ exists that expands each node s_k with $0 \leq k \leq n$. (Base case 0.) Clearly, any execution of `Append`$^*(P)$ expands s_0 from the start. (Inductive case $k \mapsto k+1$.) Suppose as inductive hypothesis that an execution expands each node in $(s_0, \ldots, s_k)$ in this order, with each node expanded after a $\pi_{j+1} = \varphi(s_j, \beta(s_j))$ for each $j < k$ satisfying $\gamma(s_j, \pi_{j+1}) = s_{j+1}$. By the above assumption, $k < n$ implies $s_k \notin \beta^\star$, so this execution calls $\varphi(s_k, \beta(s_k))$. By (1), a plan π_{k+1} exists from s_k to s_{k+1}, and by (2) π_{k+1} is a classical plan for $(\Sigma, s_k, \beta(s_k))$. By the strong completeness of $\varphi =$ `Classical`*, an execution of $\varphi(s_k, \beta(s_{k+1}))$ returns π_{k+1} and so this execution of `Append`$^*(P)$ can be extended by expanding the node s_{k+1}.

This concludes the inductive proof: this execution of `Append`$^*(P)$ expands s_n, at which point (3) implies that `Append`* terminates and returns $\pi_1. \cdots .\pi_n$. Thus, an execution of `Append`$^*(P)$ exists that returns a solution for P. Thus, `Append`* is complete for P.

($\Rightarrow$) Suppose `Append`* is complete for $P = (\Sigma, s_0, \beta)$ and that a solution for P exists. By completeness, an execution of `Append`$^*(P)$ returns a solution $\pi = \pi[1..n]$. We split the plan π in this execution of `Append`$^*(P)$ as follows:

- $\pi_0 = \varphi(s_0, \beta(s_0))$ is the value returned by the first invocation of φ, and
- $\pi_{j+1} = \varphi(s_j,\ \beta(s_j))$ is the value returned by the $(j+1)$-th invocation of φ, where $s_j = \gamma(s_0, \pi_0. \cdots .\pi_j)$ and $(\pi_0, \ldots, \pi_j)$ are the values successively returned by φ.

We prove that this sequence satisfies conditions (1)–(3) defined above in the proof of ($\Leftarrow$). Clearly, (1) $(s_j, s_{j+1}) \in R^*$ follows from the fact that $\gamma(s_j, \pi_j) = s_{j+1}$ by the soundness of `Classical`*. Also, $(s_j, s_{j+1}) \in R_\beta$ follows from the fact that $(s_j, \ldots, s_{j+1}) = \widehat{\gamma}(s_j, \pi_{j+1})$ for a plan of the form $\pi_{j+1} = \varphi(s_j, \beta(\beta(s_j))$ and the soundness of $\varphi =$ `Classical`*. Finally, $s_n = s_n^{k_n}$ follows from the fact that π is a solution leading to s_n and the soundness of `Append`*. □

Corollary 17. `Append`* *is complete for any GR problem* $P \in \mathbf{P}$ *satisfying the inclusion* $\beta^\star \subseteq (R_\beta \cap R^*)^*(s_0, \cdot)$.

Proof. Let $P = (\Sigma, s_0, \beta)$ satisfy the condition $\beta^\star \subseteq (R_\beta \cap R^*)^*(s_0, \cdot)$. Assume moreover that a solution π exists for P, as otherwise we are done. We prove that an execution of `Append`$^*(P)$ exists that returns a solution. Let $s^\star = \gamma(s_0, \pi)$ with $s^\star \in \beta^\star$. By the assumed condition, we also have that $s^\star \in (R_\beta \cap R^*)^*(s_0, \cdot)$. So there is a sequence of elements $(s_0, s_1, \ldots, s_n)$ that satisfies the conditions: (1) $(s_k, s_{k+1}) \in R^*$, (2) $(s_k, s_{k+1}) \in R_\beta$ and (3) $s_n = s^\star \in \beta^\star$. This rest of the proof is the same proof as that of Theorem 16 (completeness, $\Rightarrow$). From it, we obtain that `Append`* is complete for P. □

Lemma 18. `Append`**, as defined by Algorithm 7 with %-lines, has the termination property.*

Proof. For each invocation of φ in Algorithm 7, we know that `Classical`* has the termination property. For the nodes (goal states) directly expanded by the **while** loop in Algorithm 7, the auxiliary Algorithm keeps track of the nodes (goal states) expanded by `Append`*. Given the finite character of the search space, this implies that the **while** loop in Algorithm 7 cannot run indefinitely, and so this version of `Append`* has the termination property. □

Algorithm 8 `Replan`* (add %-lines for termination)

Input: (S, A, s_0, β)
Output: a plan π or *failure*

```
1: Let $s = s_0$ and $\pi = \emptyset$        % and $e = \{s_0\}$.
2: while $s \not\models \beta(s)$ and $\pi \neq failure$ do
3:     $\pi \leftarrow \varphi(s_0, \beta(s))$
4:     $s \leftarrow \gamma(s_0, \pi)$
5:     % Test_expanded($s$)
6: end while
7: return $\pi$
```

`Replan`*.

Lemma 19. *For any GR problem P, it holds that lfp$(\psi(P)) =$ `Replan`$^*(P)$, where ψ is as in Definition 14 and* `Replan`* *is Algorithm 8.*

Proof. The proof is analogous to that of Lemma 15 for `Append`*. One just needs to replace everywhere $\pi.\varphi(\cdot,\cdot)$ by $\varphi(\cdot,\cdot)$ and $\pi.$*failure* by *failure*. □

Theorem 20. `Replan`* *is sound but does not have the termination property.* `Replan`* *is complete for $P = (S, A, s_0, \beta)$ iff whenever a solution for P exists, then $T(s_0, \cdot) \cap \beta^\star \neq \emptyset$, where T is inductively defined by:*

$$T_0 = \mathrm{Id}_S \quad \textit{and} \quad T_{n+1} = (T_n \mid R_\beta) \cap R^* \quad \textit{and} \quad T = \textstyle\bigcup_{n\in\omega} T_n.$$

Proof. (Lack of termination.) Let P be the same counterexample in the proof of Theorem 16. Recall that $S = \{s_0, s_1\}$ has two states that are:

(i) mutually reachable $s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_0$ and (ii) goal states of each other.

Any execution of `Replan`$^*(P)$ is a succession of calls of the form $\pi_k = \varphi(s_0, \beta(s_k))$ where s_k and π_k alternate between the following values for even and resp. odd values of k:

$$s_{2k} = s_0 \qquad s_{2k+1} = s_1 \qquad \pi_{2k} = \emptyset \qquad \pi_{2k+1} = \alpha_1.$$

This sequence of calls $(\varphi(s_0, \beta(s_0)), \ldots, \varphi(s_0, \beta(s_{2k})), \varphi(s_0, \beta(s_{2k+1}), \ldots)$ never terminates in this execution of `Replan`$^*(P)$. As a consequence, `Replan`$^*(P)$ does not terminate either.

(Soundness.) The proof is analogous to that of Theorem 16. The soundness of Algorithm 8 is granted by the condition $s \models \beta(s)$ for exiting the *while* loop, which is also the terminating condition.

(Completeness, $\Rightarrow$) Suppose `Replan`* is complete for $P = (S, A, s_0, \beta)$ and assume a solution for P exists. By completeness, an execution of `Replan`$^*(P)$ outputs a solution π, say $\gamma(s_0, \pi) \in \beta^\star$. Let $(P_0, \ldots, P_n)$ be the classical problems such that a call of the form `Classical`$^*(P_k)$ is made in this execution, listed in increasing order. Let also $\pi_k = \varphi(s_0, g_k)$ be the value returned at each call and let $s_{k+1} = \gamma(s_0, \pi_k)$ (see line 4 of Algorithm 8), so that this execution of `Replan`$^*(P)$ satisfies for each $0 \leq k < n$:

$g_k = \beta(s_k)$ and $P_{k+1} = (S, A, s_0, \beta(s_k))$ and, by the soundness of φ, $\gamma(s_0, \pi_{k+1}) \models \beta(s_k)$.

By the soundness of `Replan`$^*(P)$, the returned plan π_n is a solution: $s_{n+1} \in \beta^\star$. In summary, for each $0 \leq k < n$, we have

$$(1)\ R^*(s_0, s_{k+1}) \quad \text{and} \quad (2)\ R_\beta(s_k, s_{k+1}) \quad \text{and} \quad (3)\ s_{n+1} \in \beta^\star.$$

Let us check that $(s_0, s_1, \ldots, s_{n+1})$ is a T-path. For the base case, $(s_0, s_0) \in T_0 = Id_S$. For the inductive case, assume $(s_0, s_k) \in T_k$. We prove that $(s_0, s_{k+1}) \in T_{k+1} = (T_k \mid R_\beta) \cap R^*$. Clearly, $(s_0, s_k) \in T_k$ by assumption and $(s_k, s_{k+1}) \in R_\beta)$ by (2), so we are done. The proof concludes with $(s_0, s_{n+1}) \in T_{n+1} \subseteq T$. Together with (3) $s_{n+1} \ \in \beta^\star$, we finally obtain that $s_{n+1} \in T(s_0, \cdot) \cap \beta^\star$, and so this set is not empty.

(Completeness, $\Leftarrow$) Suppose that $P = (\Sigma, s_0, \beta)$ satisfies $T(s_0,\ \cdot) \cap \beta^\star \neq \emptyset$. In case $s_0 \in \beta^\star$, any execution of `Replan`$^*(P)$ calls `Classical`$(\Sigma, s_0, \beta(s_0))$ which will return the solution $\emptyset$ and so will `Replan`$^*(P)$. Otherwise, the above condition amounts to $T_{n+1}(s_0, \cdot) \cap \beta^\star \neq \emptyset$ for some $n{+}1 > 0$. Let us choose such a non-emptyset whose index $n{+}1$ is minimal. Let then $s_{n+1} \in T_{n+1}(s_0, \cdot) \cap \beta^\star$ be a solution state. We prove by induction that for each $0 \leq k < n1$ there exists s_{k+1} satisfying:

$$(\text{i})\ (s_k, s_{k+1}) \in R_\beta \quad \text{and} \quad (\text{ii})\ (s_0, s_{k+1}) \in R^* \qquad \text{for all } 0 \leq k < n{+}1.$$

(Base Case 1.) From $s_{n+1} \in T_{n+1}(s_0, \cdot)$ and the construction of T_{n+1}, there is some s_1 such that $(s_0, s_1) \in T_1 = (Id_S | R_\beta) \cap R^*$. Clearly, (i) $(s_0, s_1) \in R_\beta$ follows from $(s_0, s_1) \in (Id_S | R_\beta) = R_\beta$. And (ii) is proved by $(s_0, s_1) \in T_1 \subseteq R^*$.

(Inductive Case $k \mapsto k{+}1$.) Suppose that there is s_k such that (i) $(s_{k-1}, s_k) \ \in R_\beta$ and (ii) $(s_0, s_k) \in R^*$. Since $(s_0, s_{k+1}) \in T_{k+1} \subseteq (T_k | R_\beta)$, there must be some state s_k satisfying $s_0\ T_k\ s_k R_\beta s_{k+1}$ and also $s_0 R^* s_{k+1}$. These immediately give us (i) and resp. (ii).

The general claims (i)–(ii) that follow from this inductive proof permit us to define the following execution of `Replan`$^*(P)$: a sequence of $n{+}1$ calls of the form $\varphi(s_0, \beta(s_k))$ (for $0 \leq k \leq n$) that return a plan π_{k+1}. Let us show that this is indeed an execution of `Replan`$^*(P)$. By the minimality of $0 < n{+}1$, we know that π_k is not a solution for any $0 \leq k \leq n$. Hence the call $\varphi(s_0, \beta(s_{k+1}))$ will be made after expanding $s_{k+1} = \gamma(s_0, \pi_k)$. At the $(n{+}1)$-th iteration of the **while** loop, the plan returned π_{n+1} is a solution, which exits the loop and then `Replan`$^*(P)$ returns π_{n+1}. Hence `Replan`* is complete for P. □

Corollary 21. `Replan`* *is complete for P whenever $\beta^\star \subseteq R^*_\beta(s_0, \cdot) \subseteq R^*(s_0, \cdot)$.*

Proof. Assume P satisfies: (1) $\beta^\star \subseteq R^*_\beta(s_0, \cdot)$ and (2) $R^*_\beta(s_0, \cdot) \subseteq R^*(s_0, \cdot)$. Assume that a solution π exists for P, and let $s^\star = s_0.\pi \in \beta^\star$. By (1), there is a sequence of states $(s_0, s_1, \ldots, s_n)$ with $s_n = s^\star$ such that $s_k R_\beta s_{k+1}$ for each $0 \leq k < n$. Without loss of generality assume moreover that s_n is minimal with the property $s_n \in \beta^\star$ among this sequence; i.e. $s_1, \ldots, s_{n-1} \notin \beta^\star$. We show by induction that there is an execution of `Replan`$^*(P)$ that expands exactly the nodes $(s_0,\ \ldots, s_n)$ and, as an auxiliary claim, that for each $0 \leq k < n$ a plan to s_{k+1} is returned, possibly by a call of the form $\varphi(s_0, \beta(s_k))$.

(Base case $k = 0$.) s_0 is expanded by any execution of `Replan`$^*(P)$, as it is the initial value of s in Algorithm 8. Let us show the auxiliary claim. First consider the case $s_0 \in \beta^\star$. Then $s^\star = s_0$ and so the above sequence is (s_0) and the empty plan $\pi = \emptyset$ is returned (lines 1–2). For the second case, assume otherwise: $s_1 \neq s_0 \notin \beta^\star$. From $s_0 R_\beta s_1$, we have

Algorithm 9 β`Saturate` (add %-line for termination)

```
Input: $(S, A, s_0, \beta)$
Output: a plan $\pi$ or failure
 1: Let $s = s_0$ and $e = \emptyset$.
 2: while $s \not\models \beta(s)$ and $s \notin e$ do
 3:     $s \leftarrow \gamma(s, \alpha(s))$
 4:     % $e \leftarrow e \cup \{s\}$
 5: end while
 6: if $s \notin e$ then
 7:     $\pi \leftarrow \varphi(s_0, s)$
 8: else
 9:     $\pi \leftarrow$ failure
10: end if
11: return $\pi$
```

$s_1 \in \beta(s_0)$. The rest of the proof of the auxiliary claim, that a plan $\pi_1 = \varphi(s_0, \beta(s_0))$ is returned, is as in the inductive case, and so we omit it.

(Inductive case $k \mapsto k+1$.) Assume that an execution of `Replan`$^*(P)$ exists that expands $s_0, \ldots, s_k$ and generates the corresponding plans $\pi_0, \ldots, \pi_k$ with $s_j = s_0.\pi_j$ (for $0 \leq j \leq k$). Again, in case $s_k \in \beta^\star$, this execution terminates with a solution π_k (line 2). Assume then otherwise: $s_k \notin \beta^\star$. From $s_0 R_\beta \ldots R_\beta s_k R_\beta s_{k+1}$ and (2), we obtain that $s_{k+1} \in R^*_\beta(s_0, \cdot) \subseteq R^*(s_0, \cdot)$, so a plan π_{k+1} exists with $s_{k+1} = s_0.\pi_{k+1}$. Since $s_k \notin \beta^\star$, a call of the form $\varphi(s_0, s_k)$ is made (lines 2–3). By Proposition 2, we can extend this execution of `Replan`$^*(P)$ so that this call returns the value $\pi_{k+1} = \varphi(s_0, s_k)$.

This completes the inductive proof. This execution of `Replan`$^*(P)$ generates thus a plan π_n with $s_n = s_0.\pi_n \in \beta^\star$, which fails the test $s_n \not\models \beta(s_n)$ (line 2), exiting the **while** loop and returning a plan π_n to a solution state s_n. Thus, `Replan`* is complete for P. □

Lemma 22. `Replan`**, as defined by Algorithm 8 with %-lines, has the termination property.*

Proof. The proof is analogous to that of Lemma 18. Simply replace everywhere in this proof `Append`$^*(P)$ by `Replan`$^*(P)$. □

β`Saturate`.

Lemma 23. *For any GR problem P, it holds that lfp$(\psi(P)) = \beta$`Saturate`(P), where ψ is as in Definition 15 and β`Saturate` is Algorithm 9.*

Proof. Observe first that Algorithm 9 (without the %-line) keeps the value $e = \emptyset$ during all the execution, and so the condition for exiting the **while** loop amounts to $s \not\models \beta(s)$, i.e. $s \notin \beta^\star$. Like Definition 15, this version then never returns $\pi =$ *failure*. The proof of the claim is by cases.

(Case $s_0 \in \beta^\star$.) Then $\psi(\emptyset) = \emptyset.\alpha(\gamma(s_0, \emptyset)) = \emptyset.\alpha(s_0) = \alpha(s_0)$. By definition, $\alpha(s_0) = (s_0, \beta(s_0))$ but because $s_0 = \beta^\star$, $\alpha(s_0)$ is an idle action (fn. **??**) and so $\psi(\emptyset) = \emptyset$. Clearly, in this case Algorithm 9 immediately exits the **while** loop and returns $\emptyset$ as well.

(Case $s_0 \notin \beta^\star$.) Starting with $\pi = \emptyset$, $\psi(\pi)$ keeps updating the (fictional) plan as follows $\pi \longmapsto \pi.\alpha(\gamma(s_0, \pi))$. Similarly, in Algorithm 9, these updates are exactly what the **while** loop does. Consider the two subcases:

(Subcase: $R^\diamond \cap \beta^\star = \emptyset$.) We prove that in this case Definition 15 enters a cycle $\pi.\alpha(s).\cdots.\alpha(s).\cdots$. The reason for this is the following: as a consistent goal function β induces a serial relation R_β, one can always extend any R_β-path one step further (maybe with a reflexive edge). In finite vocabularies At, moreover, the state space S is finite, and so by Dirichlet's box principle the iteration of $s \longmapsto \gamma(s, \alpha(s))$ eventually encounters a solution state or a cycle around some state $s_n = s_{n+k+1}$:

$$\begin{tikzcd}
s_0 \arrow[r, "\alpha(s_0)"] & \cdots \arrow[r] & s_n \arrow[dr, "\alpha(s_n)"] & \\
& s_{n+k} \arrow[ur, "\alpha(s_{n+k})"'] & & s_{n+1} \arrow[dl, "\alpha(s_{n+1})"] \\
& & \cdots \arrow[ul] &
\end{tikzcd}$$

Each finite fragment of the infinite plan $\pi.(\alpha(s_n).\cdots.\alpha(s_{n+k}))^*$ is thus generated by running the **while** loop in Algorithm 9. Hence, β`Saturate`(P) does not terminate. In Definition 15, this subcase implies that $\psi(\cdot)$ always stays in the first case $\gamma(s_0, \pi) \notin \beta^\star$, and so the same cyclic behaviour is exhibited by the infinite updates $\pi \longmapsto \pi.\alpha(\gamma(s_0, \pi))$.

(Subcase: $R^\diamond \cap \beta^\star \neq \emptyset$.) Let s be the unique element in this non-empty set. Algorithm 9 immediately exits the **while** loop with such $s \in \beta^\star$ and some solution, say π_1 with $s = \gamma(s_0, \pi_1)$, and then it returns a value of $\varphi(s_0, s)$. For Definition 15, after $|R^\diamond|$-many applications $\pi \longmapsto \psi(\pi)$ under the first case, we obtain $\psi(\cdot) = \pi_1$. Using the above fact $\gamma(s_0, \pi_1) \in \beta^\star$, together with $\pi_1 \notin A^*$, we have $\pi(\pi_1) = \varphi(s_0, \gamma(s_0, \pi))$ (second case) which will become a fixed point in the next application of $\psi(\cdot)$ (third case). □

Theorem 24. *β`Saturate` is sound but does not terminate. β`Saturate` is complete for $P = (S, A, s_0, \beta)$ iff whenever a solution exists, $R^*(s_0, \cdot) \cap \beta^\star \cap R^\diamond(s_0) \neq \emptyset$.*

Proof. (Lack of termination.) This is partly discussed in the proof of Lemma 23 (first subcase), but let us prove it with an example. Let (S, A) be defined by the two mutually desirable states $s_0 = \{p\}$ and $s_1 = \{\neg p\}$, which are also mutually reachable. That is,

$$\alpha(s_0) = (s_0, s_1) \text{ and } \alpha(s_1) = (s_1, s_0) \quad \text{ and so } \quad \{p\} \xrightarrow{\alpha(s_0)} \{\neg p\} \xrightarrow{\alpha(s_1)} \{p\}.$$

Starting with the empty plan $\emptyset$, the function $\psi(\cdot)$ keeps producing the following values:

$$\emptyset \xrightarrow{\psi} \alpha(s_0) \xrightarrow{\psi} \alpha(s_0).\alpha(s_1) \xrightarrow{\psi} \alpha(s_0).\alpha(s_1).\alpha(s_0) \xrightarrow{\psi} \ldots$$

From this, we conclude that the applications $\pi \longmapsto \psi(\pi)$ (first case) do not terminate, and so neither does $\mathit{lfp}(\psi(P))$.

(Soundness.) Let $P = (S, A, s_0, \beta)$ be a GR problem. For Algorithm 9, any plan π returned by β`Saturate`(P) results from a call of the form `Classical`(S, A, s_0, s). This state s must cause the execution of β`Saturate`(P) to exit the **while** loop (line 2) and for this, it must satisfy $s \models \beta(s)$, i.e. $s \in \beta^\star$. By the soundness of `Classical`, $\gamma(s_0, \pi) = s$, and so π is a solution for P.

(Completeness, $\Leftarrow$.) Let $P = (S, A, s_0, \beta)$ satisfy the righthand condition, and moreover assume that a solution exists for P, so that $R^*(s_0, \cdot) \cap \beta^\star \cap R^\diamond_\beta(s_0) \neq \emptyset$. Let then $s^\star$ be the unique element of this non-empty set. From $s^\star \in R^*(s_0, \cdot)$, (1) there is a plan $\pi \in A^*$ satisfying $\gamma(s_0, \pi) = s^\star$ and, without loss of generality, further assume that such a plan does not traverse any node twice. From the fact that $s^\star \in \beta^\star$, any execution of β`Saturate`(P) expanding this node will exit the **while** loop. Finally, from $s^\star \in R^\diamond_\beta(s_0)$ we know that $s^\star$ obtains from s_0 after finitely-many updates with actions $\alpha(s) \in A'$. Algorithm 9 will exit the **while** loop when it expands node $s^\star$ and proceed to call φ. Combining (1), the non-redundancy of π and the completeness of `Classical`, an execution of `Classical`$(S, A, s_0, s^\star)$ exists that returns π. In this execution, π will also be the output returned by this execution of β`Saturate`(P). Hence, β`Saturate` is complete for P.

(Completeness, $\Rightarrow$.) Assume that β`Saturate` is complete for $P = (S, A, s_0, \beta)$ and moreover that a solution exists for this problem P. Hence, a solution π is returned by some execution of β`Saturate`(Σ, s_0, β). Let $s^\star = \gamma(s_0, \pi)$. An inspection of Algorithm 9 shows that π must be the classical plan returned by `Classical`$(S, A, s_0, s^\star)$ and also that $s^\star$ must result from a finite number n of updates with actions $\alpha \in A'$. We prove by cases that the righthand condition holds.
(Case $n = 0$.) Then no such updates occur for s_0 to reach $s^\star$. That is, $s_0 = s^\star$ and so $s_0 \in \beta^\star$, from which we conclude that the call returns the plan $\pi = \emptyset$. This case immediately verifies: $s_0 \in R^*(s_0, \cdot)$, and also $s_0 \in \beta^\star$ and, finally, $s_0 = \gamma(s_0, \alpha(s_0)) \in R^\diamond(s_0)$. That is, $s_0 \in R^*(s_0, \cdot) \cap \beta^\star \cap R^\diamond(s_0)$. Hence this set is non-empty.
(Case $n > 0$.) Let $s_1 = \gamma(s_0, \alpha(s_0))$. An easy induction shows that a sequence $(s_1, \ldots, s_n)$ exists such that for each $0 < k < n$,

$$s_{k+1} = \gamma(s_k, \alpha(s_k)) \models \beta(s_k) \quad \text{and so} \quad s_{k+1} \in R^\diamond(s_0) \text{ provided that } s_k \in R^\diamond(s_0).$$

From this, we conclude that $s^\star = s_n \in R^\diamond(s_0)$. Since β`Saturate`(P) returns π, we also have $s_n \in R^*(s_0, \cdot)$. Finally. $s_n = s^\star \in \beta^\star$. Putting these facts together, it follows that $s_n \in R^*(s_0, \cdot) \cap \beta^\star \cap R^\diamond(s_0) \neq \emptyset$. □

Lemma 25. β`Saturate`*, defined by Algorithm 9 with %-lines, has the termination property.*

Proof. Algorithm 9 in %-line 4 maintains a list e of nodes already expanded during the **while** loop. Since the search space is finite, iterating the $s \longmapsto \alpha(s)$ will eventually reach a solution state or a node already visited. Any execution of β`Saturate`(P) thus eventually exits the **while** loop and return a classical plan or *failure* (in the former case) or *failure* in the latter. □

A Hierarchy of Completeness Classes

Proposition 26. *The classes of GR problems for which Algorithms 4–9 are complete can be arranged as follows:*

$$\beta\texttt{Classical}^* = \texttt{UniClass}^* \subset \texttt{Append}^* \subset \texttt{Replan}^* \subset \texttt{Universal}$$

and β`Saturate` $\subset$ `Universal`*.* β`Saturate` *is incomparable with all other planners.*

Proof. We show that: (1) each inclusion $\subseteq$ holds, (2) the corresponding inclusions are proper $\subset$ and finally, (3) the incomparability claims.

(1. Inclusions $\subseteq$.)

($\beta\texttt{Classical}^* \subseteq \texttt{UniClass}^*$.) Corollary 12 showed that $\texttt{UniClass}^*(P) \supseteq \beta\texttt{Classical}^*(P) \cap \texttt{Universal}(P)$, for any $P \in \mathbf{P}$. From this, it is immediate that $\beta\texttt{Classical}^*(P) \subseteq \texttt{UniClass}^*(P)$. Assume that $\beta\texttt{Classical}^*$ is complete for P and that a solution for P exists, so that a solution is returned by an execution of $\beta\texttt{Classical}^*(P)$. By the previous inclusion, the same solution is also returned by an execution of $\texttt{UniClass}^*(P)$, and so we are done.

($\texttt{UniClass}^* \subseteq \beta\texttt{Classical}^*$.) Suppose that an execution of $\texttt{UniClass}^*(P)$ returns a solution π. A quick look at Algorithms 4, 6 shows that an execution of $\beta\texttt{Classical}^*(P)$ also generates the same plan π. From the execution of $\texttt{UniClass}^*(P)$, we know that π reaches the goal state $\beta(s_0)$. Because of this, the execution of $\beta\texttt{Classical}^*(P)$ that generates π also terminates and returns π. Hence $\beta\texttt{Classical}^*$ is complete for P.

($\texttt{UniClass}^* \subseteq \texttt{Append}^*$.) Any execution of $\texttt{UniClass}^*(P)$ that returns a solution π is essentially an execution of $\texttt{Append}^*(P)$ that exits the **while** loop after the first call to φ. This execution returns the same solution π.

($\texttt{Append}^* \subseteq \texttt{Replan}^*$.) Let an execution of $\texttt{Append}^*(P)$ return a solution π, with π generated as a sequence of subplans $\pi = \pi_1.\pi_2.\cdots.\pi_n$. (That is, for π_k being the value returned by the call to φ in the k-th iteration of Algorithm 7's **while** loop.) Consider the plans $\pi'_k = \pi_1.\cdots.\pi_k$ for each $1 \leq k \leq n$. An execution of $\texttt{Replan}^*(P)$ exists that generates the value π'_k at the k-th iteration of Algorithm 8's **while** loop. Clearly, this execution of $\texttt{Replan}^*(P)$ returns $\pi'_n = \pi$, which is a solution. Hence, $\texttt{Replan}^*$ is also complete for P.

(All $\texttt{OffGR}$ planners $\subseteq \texttt{Universal}$). This simply follows from $\mathbf{P}$ being the class of all $\texttt{OffGR}$ problems and, at the same time, the completeness class of $\texttt{Universal}$ (Lemma 9).

(2. Proper inclusions $\subset$.) We refer to the examples discussed to prove that all the $\subseteq$ inclusions shown above are indeed proper $\subset$.

($\texttt{Classical} \subset \texttt{UniClass}^*$)	Ex. 9	($\texttt{Append}^* \subset \texttt{Universal}$)	Ex. 13
($\texttt{UniClass}^* \subset \texttt{Append}^*$)	Ex. 11	($\texttt{Replan}^* \subset \texttt{Universal}$)	Ex. 13
($\texttt{Append}^* \subset \texttt{Replan}^*$)	Ex. 12	($\beta\texttt{Saturate} \subset \texttt{Universal}$)	Ex. 14

(3. Incomparability.) The incomparability claims are all shown by the next pair of examples:

$$(\beta\texttt{Saturate} \not\subseteq, \not\supseteq \{\beta\texttt{Classical}^*, \texttt{UniClass}^*, \texttt{Append}^*, \texttt{Replan}^*\} \quad \text{Ex. 13, Ex. 14.}$$ □

Proposition 27. *For each* $\texttt{Planner} \in \{\texttt{UniClass}, \texttt{Append}, \texttt{Replan}\}$*, its completeness class satisfies:* $\texttt{Planner} \subset \texttt{Planner}^*$.

Proof. Let $\texttt{Planner} \in \{\texttt{UniClass}, \texttt{Append}, \texttt{Replan}\}$.

Let us first show the inclusion $\texttt{Planner} \subseteq \texttt{Planner}^*$. Let P be an arbitrary GR problem. Clearly, any execution of $\texttt{Planner}(P)$ that returns a plan π corresponds to an execution of $\texttt{Planner}^*(P)$, namely one in which all invocations of $\texttt{Choice}$ (Algorithm 2) returns $\texttt{true}$. As a consequence, this execution of $\texttt{Planner}^*(P)$ assigns the same value π after each call to φ. Since this applies in particular to solution plans, whenever $\texttt{Planner}(P)$ may return a solution, so does $\texttt{Planner}^*(P)$.

Finally, Example 16 showed that these inclusions are proper: $\texttt{Planner} \subset \texttt{Planner}^*$. □

Proposition 28. *The completeness classes of* $\{\texttt{UniClass}, \texttt{Append}, \texttt{Replan}\}$ *arrange as:*

$$\texttt{UniClass} \subset \{\texttt{Append}, \texttt{Replan}\} \subset \texttt{Universal}.$$

The completeness classes of Append *and* Replan *are incomparable.*

Proof. Each inclusion $\subseteq$ is shown by a similar proof to that of Proposition 26, here omitted. The same examples used in the proof of Proposition 26 show that these inclusions are proper and also one direction of the incomparability claim; for the other, we use Example 17 above:

$(\texttt{UniClass} \subset \texttt{Append})$	Ex. 11	$(\texttt{Append} \not\subseteq \texttt{Replan})$	Ex. 17
$(\texttt{Append} \subset \texttt{Universal})$	Ex. 13	$(\texttt{Replan} \not\subseteq \texttt{Append})$	Ex. 12
$(\texttt{Replan} \subset \texttt{Universal})$	Ex. 13		

□

On the Reducibility of OffGR to Classical Planning

Lemma 29. *Let* $P = (S, A, s_0, \beta_0)$ *be a GR problem, with* β_0 *induced by* $(\mathcal{G}, \vdash_0)$*. Then,*

$$s \in \beta^\star \quad \textit{iff} \quad s \models \{(\neg\ell \vee \ell')\}_{(\ell,\ell')\in\mathcal{G}}.$$

Proof. We use the following equivalences:

$$\begin{array}{lll}
 & s \models \{(\neg\ell \vee \ell')\}_{(\ell,\ell')\in\mathcal{G}} & \\
\text{iff} & s \models \neg\ell \vee \ell' \qquad \text{for all } (\ell,\ell') \in \mathcal{G} & \\
\text{iff} & s \models \ell \;\Rightarrow\; s \models \ell' \qquad \text{for all } (\ell,\ell') \in \mathcal{G} & \text{(semantics)} \\
\text{iff} & s \models \{\ell' : (\ell,\ell') \in \mathcal{G}, s \models \ell\} & \\
\text{iff} & s \models \beta(s) & (\text{def. } \beta \text{ under } \vdash_0) \\
\text{iff} & s \in \beta^\star & (\text{def. } \beta^\star).
\end{array}$$

□

Proposition 30 (Compilation A)**.** *For any* $P = (S, A, s_0, \beta_0)$*, the map* $P \longmapsto P'$ *in Definition 22 is a compilation for* β_0 *satisfying, moreover,* $\texttt{Planner}_{\rhd T}(P') \upharpoonright A \subseteq \texttt{UniClass}^*(P)$*.*

Proof. Let $\pi' \in \texttt{Planner}_{\rhd T}(P')$ be arbitrary. Clearly, π' is of the form $\pi' = \pi.\texttt{end}i$ for some $\texttt{end}i \in A'$ and with $\pi \in A^*$. The reason is that π' solves P' so that $\gamma(s'_0, \pi') \models \mathit{final}$. By the plan minimality of $\pi \in \texttt{Planner}_{\rhd T}(P')$, no other A' action can occur in π, and so $\pi' \upharpoonright A = \pi$.

($\texttt{Planner}_{\rhd T}(P') \upharpoonright A \subseteq \beta\texttt{Classical}^*(P)$.) Let us show first that $\pi \in \beta\texttt{Classical}^*(P)$. Since $\mathit{pre}(\texttt{end}i) = \beta(s_0)$, we have that $\gamma(s_0, \pi) \models \beta(s_0)$. By Lemma 7, $\beta\texttt{Classical}^*(P) = \texttt{Classical}^*(P'')$ where $P'' = (S, A, s_0, \beta_0(s_0))$. By Proposition 2, $\texttt{Classical}^*$ is strongly complete, and so $\pi \in \texttt{Classical}^*(P'') = \beta\texttt{Classical}^*(P)$.

($P \longmapsto P'$ is a compilation for β_0.) It suffices to prove that $\pi = \pi' \upharpoonright A$ satisfies $\gamma(s_0, \pi) \in \beta^\star$. In the previous claim, we showed that $\gamma(s_0, \pi) \models \beta(s_0)$. That is, for any norm $(\ell, \ell') \in \mathcal{G}$, (1) if $\ell' \in \beta(s_0)$ then $\gamma(s_0, \pi) \models \ell'$. On the other hand, (2) for each other norm $(\ell, \ell') \in \mathcal{G}$ with $\ell' \notin \beta(s_0)$, the definition of $\mathit{eff}(\texttt{end}i)$ gives: either $\gamma(s_0, \pi) \models \neg\ell$ or $\gamma(s_0, \pi) \models \mathit{goal}(\ell'), \ell'$. Combining (1) and (2), we obtain that $\gamma(s, \pi) \models \neg\ell \vee \ell'$ for each norm $(\ell, \ell') \in \mathcal{G}$. By Lemma 29, we conclude that $\gamma(s_0, \pi) \in \beta^\star$. Hence, $\pi = \pi' \upharpoonright A$ is a solution for P, and so Definition 22 is a compilation for β_0.

($\texttt{Planner}_{\rhd T}(P') \upharpoonright A \subseteq \texttt{UniClass}^*(P)$.) It follows from the last two claims that $\pi \in \beta\texttt{Classical}^*(P)$ and $s_0.\pi \in \beta^\star$. By Definitions 10–12, it holds that $\pi \in \texttt{UniClass}^*(P)$. □

Proposition 31 (Compilation B)**.** *For any $P = (\Sigma, s_0, \beta_0)$, the map $P \longmapsto P'$ in Definition 23 is a compilation for β_0 satisfying, moreover,* $\texttt{Planner}_{\triangleright T}(P') \restriction A = \texttt{Universal}(P)$*.*

Proof. As in the proof of Proposition 30, we have that any output $\pi' \in \texttt{Planner}_{\triangleright T}(P')$ is of the form $\pi' = \pi.\texttt{end}i$ with $\pi \in A^*$.

($P \longmapsto P'$ is a compilation for β_0.) For each norm $(\ell, \ell') \in \mathcal{G}$, the definition of $\mathit{eff}(\texttt{end}i)$ gives: either $\gamma(s_0, \pi) \models \neg\ell$ or $\gamma(s_0, \pi) \models \mathit{goal}(\ell'), \ell'$. Hence, $\gamma(s, \pi) \models \neg\ell \vee \ell'$ for each norm $(\ell, \ell') \in \mathcal{G}$. By Lemma 29, we conclude that $\gamma(s_0, \pi) \in \beta^\star$. Hence, $\pi = \pi' \restriction A$ is a solution for P.

($\texttt{Planner}_{\triangleright T}(P') \restriction A \subseteq \texttt{Universal}(P)$.) We just showed that π is a solution for P. By Remark 3, moreover, no subplan π_1 of π extends into a plan $\pi.\texttt{end}j$ solving P', i.e. satisfying $\gamma(s'_0, \pi.\texttt{end}j) \not\models \mathit{final}$. Hence, no such subplan $\pi_1 \in A^*$ satisfies $(\neg\ell \vee \ell')$ for all norms $(\ell, \ell') \in \mathcal{G}$. By Lemma 29, we conclude that $\gamma(s_0, \pi_1) \notin \beta^\star$. A quick look at Algorithm 5 shows that an execution of $\texttt{Universal}(P)$ exists that returns π.

($\texttt{Planner}_{\triangleright T}(P') \restriction A \supseteq \texttt{Universal}(P)$.) This inclusion is proved by reading the last proof backwards: the minimality of plans for P' and solutions for P is preserved in both directions, and Lemma 29 states an equivalence between *no norm violations* and $\beta^\star$-membership. □

References

Aha, D. W. (2018). Goal reasoning: Foundations, emerging applications, and prospects. *AI Mag.*, *39*(2), 3–24.

Anderson, M., & Anderson, S. L. (2014). GenEth: A general ethical dilemma analyzer. In *Proceedings of the 28th AAAI Conference on AI*, pp. 253–261.

Baier, J. A., & McIlraith, S. A. (2008). Planning with preferences. *AI Mag.*, *29*(4), 25–36.

Baum, S. D. (2020). Social choice ethics in artificial intelligence. *AI Soc.*, *35*(1), 165–176.

Behnke, G., Speck, D., Katz, M., & Sohrabi, S. (2023). On partial satisfaction planning with total-order htns. In Koenig, S., Stern, R., & Vallati, M. (Eds.), *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling, July 8-13, 2023, Prague, Czech Republic*, pp. 42–51. AAAI Press.

Bonassi, L., De Giacomo, G., Favorito, M., Fuggitti, F., Gerevini, A. E., & Scala, E. (2023). Planning for temporally extended goals in pure-past linear temporal logic. In Koenig, S., Stern, R., & Vallati, M. (Eds.), *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling, Prague, Czech Republic, July 8-13, 2023*, pp. 61–69. AAAI Press.

Brafman, R. I., & Domshlak, C. (2009). Preference handling - an introductory tutorial. *AI Mag.*, *30*(1), 58–86.

Bratman, M. E. (1991). *Intention, Plans, and Practical Reason.* David Hume Series. Center for the Study of Language and Information.

Braziunas, D., & Boutilier, C. (2007). Minimax regret based elicitation of generalized additive utilities. In Parr, R., & van der Gaag, L. C. (Eds.), *UAI 2007, Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence, Vancouver, BC, Canada, July 19-22, 2007*, pp. 25–32. AUAI Press.

Brewka, G. (1989). Preferred subtheories: An extended logical framework for default reasoning. In Sridharan, N. S. (Ed.), *Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989*, pp. 1043–1048. Morgan Kaufmann.

Brewka, G. (2004). A rank based description language for qualitative preferences. In de Mántaras, R. L., & Saitta, L. (Eds.), *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pp. 303–307. IOS Press.

Brewka, G., & Eiter, T. (1999). Preferred answer sets for extended logic programs. *Artificial Intelligence*, *109*, 297–356.

Brewka, G., & Eiter, T. (2000). Prioritizing default logic. In *Intellectics and computational logic*, pp. 27–45. Springer.

Broersen, J. M., Dastani, M., & van der Torre, L. (2005). Beliefs, obligations, intentions, and desires as components in an agent architecture. *Int. J. Intell. Syst.*, *20*(9), 893–919.

Carmo, J., & Jones, A. J. I. (2002). Deontic logic and contrary-to-duties. In Gabbay, D., & Guenthner, F. (Eds.), *Handbook of Philosophical Logic (2nd edition)*, Vol. 8, pp. 265–343. Kluwer Academic Publishers.

Carmo, J., & Jones, A. J. I. (2013). Completeness and decidability results for a logic of contrary-to-duty conditionals. *J. Log. Comput.*, *23*(3), 585–626.

Chellas, B. F. (1980). *Modal Logic: An Introduction.* Cambridge University Press.

Chien, S. A., Knight, R., Stechert, A., Sherwood, R., & Rabideau, G. (2000). Using iterative repair to improve the responsiveness of planning and scheduling. In Chien, S. A., Kambhampati, S., & Knoblock, C. A. (Eds.), *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO, USA, April 14-17, 2000*, pp. 300–307. AAAI.

Chisholm, R. M. (1963). Contrary-to-duty imperatives and deontic logic. *Analysis*, *24*(2), 33–36.

Cox, M. T. (2007). Perpetual self-aware cognitive agents. *AI Mag.*, *28*(1), 32–46.

Cox, M. T. (2017). A model of planning, action, and interpretation with goal reasoning. *Advances in Cognitive Systems*, *5*, 57–76.

Doyle, J., Shoham, Y., & Wellman, M. P. (1991). A logic of relative desire (preliminary report). In Ras, Z. W., & Zemankova, M. (Eds.), *Methodologies for Intelligent Systems, 6th International Symposium, ISMIS '91, Charlotte, N.C., USA, October 16-19, 1991, Proceedings*, Vol. 542 of *Lecture Notes in Computer Science*, pp. 16–31. Springer.

Dyrkolbotn, S., Pedersen, T., & Slavkovik, M. (2018). On the distinction between implicit and explicit ethical agency. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society, AIES 2018, New Orleans, LA, USA, February 02-03, 2018*, pp. 74–80.

Fischer, M. J., & Ladner, R. E. (1979). Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, *18*, 194–211.

Gabbay, D., J. Horty, X. P., van der Meyden, R., & van der Torre, L. (Eds.). (2013). *Handbook of Deontic Logic and Normative Systems*. College Publications.

Ghallab, M., Nau, D. S., & Traverso, P. (2004). *Automated planning - theory and practice*. Elsevier.

Ghallab, M., Nau, D. S., & Traverso, P. (2016). *Automated Planning and Acting*. Cambridge University Press.

Goble, L. (2005). A logic for deontic dilemmas. *J. Applied Logic*, *3*(3-4), 461–483.

Governatori, G., Rotolo, A., & Riveret, R. (2018). A deontic argumentation framework based on deontic defeasible logic. In Miller, T., Oren, N., Sakurai, Y., Noda, I., Savarimuthu, B. T. R., & Son, T. C. (Eds.), *PRIMA 2018: Principles and Practice of Multi-Agent Systems - 21st International Conference, Tokyo, Japan, October 29 - November 2, 2018, Proceedings*, Vol. 11224 of *Lecture Notes in Computer Science*, pp. 484–492. Springer.

Haddawy, P., Ha, V. A., Restificar, A. C., Geisler, B., & Miyamoto, J. (2003). Preference elicitation via theory refinement. *J. Mach. Learn. Res.*, *4*, 317–337.

Hansen, J. (2008). Prioritized conditional imperatives: problems and a new proposal. *Autonomous Agents and Multi-Agent Systems*, *17*(1), 11–35.

Hansson, B. (1969). An analysis of some deontic logics. *Nous*, *3*(4), 373–398.

Hansson, S. O. (2013). Alternative semantics for deontic logic. In Gabbay et al. (Gabbay et al., 2013), pp. 445–497.

Haslum, P., Lipovetzky, N., Magazzeni, D., & Muise, C. (2019). *An Introduction to the Planning Domain Definition Language*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Hawes, N. (2011). A survey of motivation frameworks for intelligent systems. *Artif. Intell.*, *175*(5-6), 1020–1036.

Horty, J. (2007). Defaults with priorities. *Journal of Philosophical Logic*, *36*, 367–413.

Horty, J. (2012). *Reasons as Defaults*. Oxford University Press.

Jaidee, U., Muñoz-Avila, H., & Aha, D. W. (2011). Integrated learning for goal-driven autonomy. In Walsh, T. (Ed.), *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pp. 2450–2455. IJCAI/AAAI.

Katz, M. (2019). Red-black heuristics for planning tasks with conditional effects. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pp. 7619–7626. AAAI Press.

Kautz, H. A., & Selman, B. (1991). Hard problems for simple default logics. *Artif. Intell.*, *49*(1-3), 243–279.

Klenk, M., Molineaux, M., & Aha, D. W. (2013). Goal-driven autonomy for responding to unexpected events in strategy simulations. *Comput. Intell.*, *29*(2), 187–206.

Knoblock, C. A. (1995). Planning, executing, sensing, and replanning for information gathering. In *Proceedings of the Fourteenth International Joint Conference on Artificial*

Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes, pp. 1686–1693. Morgan Kaufmann.

Liao, B., Oren, N., van der Torre, L., & Villata, S. (2016). Prioritized norms and defaults in formal argumentation. In Roy, O., Tamminga, A., & Willer, M. (Eds.), *Proceedings of the 13th International Conference on Deontic Logic and Normative Systems (DEON 2016)*, pp. 139–154. College Publications.

Liao, B., Pardo, P., Slavkovik, M., & van der Torre, L. (2023). The jiminy advisor: Moral agreements among stakeholders based on norms and argumentation. *Journal of Artificial Intelligence Research*, *77*, 737–792.

Loewer, B., & Belzer, M. (1983). Dyadic deontic detachment. *Synthese*, *54*, 295–318.

Makinson, D. (1999). On a fundamental problem of deontic logic. In McNamara, P., & Prakken, H. (Eds.), *Norms, Logics and Information Systems. New Studies in Deontic Logic and Computer Science*, Vol. 49 of *Frontiers in Artificial Intelligence and Applications*, pp. 29–53. IOS Press.

Meneguzzi, F., & De Silva, L. (2015). Planning in bdi agents: a survey of the integration of planning algorithms and agent reasoning. *The Knowledge Engineering Review*, *30*, 1–44.

Meneguzzi, F., Rodrigues, O., Oren, N., Vasconcelos, W. W., & Luck, M. (2015). BDI reasoning with normative considerations. *Eng. Appl. Artif. Intell.*, *43*, 127–146.

Meyer, J.-J. C., Broersen, J., & Herzig, A. (2015). Bdi logics. In van Ditmarsch, H., Halpern, J., van der Hoek, W., & Kooi, B. (Eds.), *Handbook of Logics for Knowledge and Belief*, chap. 10, pp. 453–498. College Publications.

Modgil, S., & Prakken, H. (2013). A general account of argumentation with preferences. *Artif. Intell.*, *195*, 361–397.

Moor, J. H. (2006). The nature, importance, and difficulty of machine ethics. *IEEE Intelligent Systems*, *21*(4), 18–21.

Nebel, B. (2000). On the compilability and expressive power of propositional planning formalisms. *J. Artif. Intell. Res.*, *12*, 271–315.

Paisner, M., Cox, M. T., Maynord, M., & Perlis, D. (2014). Goal-driven autonomy for cognitive systems. In Bello, P., Guarini, M., McShane, M., & Scassellati, B. (Eds.), *Proceedings of the 36th Annual Meeting of the Cognitive Science Society, CogSci 2014, Quebec City, Canada, July 23-26, 2014*. cognitivesciencesociety.org.

Parent, X., & van der Torre, L. (2018). *Introduction to Deontic Logic and Normative Systems*. College Publications.

Pigozzi, G., & van der Torre, L. (2018). Arguing about constitutive and regulative norms. *J. Appl. Non Class. Logics*, *28*(2-3), 189–217.

Prakken, H., & Sergot, M. (1997). Dyadic deontic logic and contrary-to-duty obligations. In Nute, D. (Ed.), *Defeasible deontic logic*, pp. 223–262. Kluwer Academic, Dordrecht-Boston.

Rao, A. S., & Georgeff, M. P. (1991). Modeling rational agents within a bdi-architecture. In Allen, J. F., Fikes, R., & Sandewall, E. (Eds.), *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91). Cambridge, MA, USA, April 22-25, 1991*, pp. 473–484. Morgan Kaufmann.

Ross, A. (1941). Imperatives and logic. *Theoria*, *7*, 53–71.

Shaparau, D., Pistore, M., & Traverso, P. (2006). Contingent planning with goal preferences. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pp. 927–935.

Shoham, Y. (2009). Logical theories of intention and the database perspective. *J. Philos. Log.*, *38*(6), 633–647.

Smith, D. E. (2004). Choosing objectives in over-subscription planning. In Zilberstein, S., Koehler, J., & Koenig, S. (Eds.), *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*, pp. 393–401. AAAI.

Straßer, C. (2011). A deontic logic framework allowing for factual detachment. *J. Applied Logic*, *9*(1), 61–80.

Thangarajah, J., & Padgham, L. (2011). Computationally effective reasoning about goal interactions. *J. Autom. Reason.*, *47*(1), 17–56.

Van De Putte, F., Beirlaen, M., & Meheus, J. (2019). Adaptive deontic logics. *Handbook of Deontic Logic and Normative Systems*, *2*, 367–459.

van den Briel, M., Nigenda, R. S., Do, M. B., & Kambhampati, S. (2004). Effective approaches for partial satisfaction (over-subscription) planning. In McGuinness, D. L., & Ferguson, G. (Eds.), *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, pp. 562–569. AAAI Press / The MIT Press.

van der Krogt, R., & de Weerdt, M. (2005). Plan repair as an extension of planning. In Biundo, S., Myers, K. L., & Rajan, K. (Eds.), *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), June 5-10 2005, Monterey, California, USA*, pp. 161–170. AAAI.

van Der Torre, L., & Tan, Y.-H. (1995). Cancelling and overshadowing two types of defeasibility in defeasible deontic logic. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pp. 1525–1532, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

van Zee, M., Doder, D., van der Torre, L., Dastani, M., Icard III, T. F., & Pacuit, E. (2020). Intention as commitment toward time. *Artif. Intell.*, *283*, 103270.