Re-examining the Planning Basis of Goal-driven Autonomy Problems

Rogelio E. Cardona-Rivera,[1] M. Gardone,[1] Logan Peterson,[1] Laura M. Hiatt,[2] Mark Roberts[2]
[1]Laboratory for Quantitative Experience Design
School of Computing and the Entertainment Arts and Engineering Program, University of Utah, Salt Lake City, UT, USA
[2]Navy Center for Applied Research in AI
Naval Research Laboratory, Washington, DC, USA
{rogelio@cs., m.gardone@cs., logan.peterson@}utah.edu, {laura.hiatt, mark.roberts}@nrl.navy.mil

Abstract

The study of goal-reasoning agents capable of integrated action and execution has received a great deal of attention in recent years. While practical implementations and theoretical insights of such agents have provided a wealth of flexible behavior in a variety of task environments, they tend to focus on complex environments that are far from classical planning assumptions. This paper formalizes classical planning problems where an agent can change its goal(s) during execution. We identify the minimal changes to classical planning and formalize a model that supports "classical goal reasoning."

1 Introduction

Allowing an agent to modify its goals during execution challenges several core assumptions of the classical planning model. It is perhaps unsurprising that most studies in *goal reasoning* (also called *goal-driven autonomy*) have favored environments that relax the classical assumptions by being *partially-observable*, *non-deterministic* (including those with exogenous *events*) or being *open worlds* (Aha 2018). However, there remain problems that are very close to classical ones, especially for deterministic and fully-observable environments.

Several scholars have developed formal theories of goal reasoning systems. For instance, Hawes (2011) distilled four reasons why an agent might exhibit goal reasoning, calling for their formalization. Later, Harland et al. (2014) formalized an operational semantics for a Belief-Desire-Intention (BDI) agent architecture capable of a vast array of goal reasoning mechanisms. More recently, Dannenhauer, Muñoz-Avila, and Cox (2021) formalized a comprehensive theory of how goal-driven agents might reformulate goals based on violated expectations. While these models have significantly advanced our understanding of goal-driven autonomy, their studies began with a planning model that was significantly more complex than classical planning.

There remains a gap somewhere between the full set of classical assumptions (that would only require a classical planner to solve) and the complexity of typical problems studied in goal reasoning research. In this paper we fill that gap by investigating the minimal classical assumptions that must be relaxed to arrive at "classical goal reasoning."

Contributions We found that we are able to parsimoniously describe classical goal reasoning via what we term a *goal transition system*—a novel structure that explicitly represents the hypergraph induced by goals over a classical planning problem's state-transition system. Our other contributions include:

- Formalizing the distinction between a *closed* (goal) problem, typical of classical planning, and an *open* problem that requires goal reasoning.
- Codifying which classical assumptions must change to support the definition of classical open problems.
- Sketching a goal reasoning system-agnostic language for syntactically defining goal reasoning problems that we call GRUPS: the Goal Reasoner $\cup$ Problem Solver.
- Discussing our motivating use-case for classical goal reasoning via a GRUPS-based specification.

A Motivating Example: Adventure Games Consider the task of designing an AI capable of playing *adventure games*, a kind of interactive narrative experience wherein the protagonist (typically, a human "player") drives the development of an unfolding plot (Riedl and Bulitko 2013). Canonically, these storygames present a verb-based command interface with 9 template commands as in Figure 1. Players can OPEN unlocked doors, PICK UP co-located items, DROP items from the inventory, CLOSE unlocked open doors, LOOK AT objects, GO TO cardinally-adjacent locations and through open doors, GIVE inventory items and TALK TO co-located characters, and USE inventory items on other objects (e.g. use keys on locked doors to unlock them).

As a task domain, adventure games have a finite set of states and are deterministic. The actions that players can perform (all instances of the template commands) execute instantaneously. The virtual world extends beyond what is visible in Figure 1's interface window, but the standard convention of adventure games is that the plot's time-flow is halted to invite attention on the interplay of elements within a limited area of spacetime (Black 2012). Thus, players expect that by default nothing changes beyond what they immediately see or do, making the environment effectively fully observable and static.

At first glance, all we might need to rely on for the AI's architecture is a classical planner like STRIPS (Fikes and Nils-

Figure 1: Screenshot from a canonical adventure game, with a verb-based command interface. Although classical planners could ostensibly play these games, the lack of formal semantics for *changing goals* precludes them from doing so in a systematic manner.

son 1971) coupled with a pipelined machine scheduler, especially because the main thing to *do* in these games are *quests*—i.e. executing (any sequence of) actions to establish conditions that are *distinguished* by the game due to their narrative import (Cardona-Rivera, Zagal, and Debus 2020). But despite this task environment meeting nearly all assumptions[1] that define the classical model, off-the-shelf classical planners would fail to meet this challenge due to their inability to systematically handle a change in goals.

This kind of problem requires the ability to perform goal reasoning. However, (as mentioned) agents that exhibit this kind of goal-driven autonomy today are built for partially-observable and dynamic environments, based on the tacit assumption that these features are intrinsic to the class of goal reasoning problems.

While modern goal-driven agents would undoubtedly succeed at playing an adventure game, they are mismatched to the task environment at hand. With an *excess* of capability on one hand, and a *lack* of capability within classical planning on the other, we are left with the question: might there be a middle ground? We explore an answer to that question by critically unpacking the planning basis that goal-driven autonomy was built upon.

2 Related Work

Goal-driven autonomy is a conceptual model, invented to facilitate developing agents that exhibit goal reasoning dynamics—the adoption, (re)formulation, and abandonment of goals—as a key part of the principles that govern their behavior (Molineaux, Klenk, and Aha 2010). It emerged as an extension of the *dynamic planning model* by Ghallab, Nau, and Traverso (2004), which itself describes agents capable of generating goal-directed behavior from a *given* goal in terms of three elements: (1) the **system**, representing the agent's task environment, (2) a **planner**, the agent component responsible for organizing behavior toward satisfying the goal, and (3) a **controller**, the agent component responsible for manifesting the behavior in the system.

Today, scholars consider *dynamic planning* as one of the four capabilities that *distinguish* goal-driven behavior from other forms of autonomy (Aha 2018). The other three defining capabilities are the *explicit representation*, *generation*, and *selection* of goals (Hawes 2011; Dannenhauer, Muñoz-Avila, and Cox 2021). However, work in this area has proceeded with a conflicted understanding of what *problem* goal-driven autonomy is intended to solve. This conflict has manifested in three ways that threaten systematic progress in research on goal-driven autonomy (GDA).

First, we cannot yet precisely state what *features* of a problem would motivate the use of GDA. To date, GDA problems are specified within particular goal reasoning architectures, each handling unique execution-level concerns for the agent. As surveyed by Vattam et al. (2013), these concerns include the discovery of new actions, detection of higher-utility opportunities, violation of expectations, and graceful degradation of performance, among several others. But despite the array of capabilities, we have yet to articulate a common ground that would allow us to distinguish situations in which the problem at hand calls for *revision of the goal being pursued* as opposed to *plan repair* or *re-planning*. Relatedly, we are unable to specify GDA problems in a domain-independent way—unlike dynamic planning ones, which benefit from several planner-agnostic languages like PDDL (McDermott et al. 1998).

Second, we cannot yet precisely state what a GDA problem *represents*. GDA problems have been framed as a computational model of *problem recognition*—the human cognitive process that articulates *which* goals need solving (Cox 2020). Confusingly, all of GDA's defining capabilities can manifest within dynamic planning *alone*. This is because a planner is responsible for ensuring that an action's preconditions are satisfied and not undone from the moment they are established until they are needed. Every action with unsatisfied preconditions affords the planner's *generation* of (sub)goals, and the planner must perform a *selection* of which goals to satisfy first (potentially impacting the time needed to satisfy them all). While classical planners need not reify goals to perform their function, *plan space* planners *do*—albeit in terms of flaws that must be refined via fixes (cf. Weld 1994; Kambhampati, Knoblock, and Yang 1995). Is this goal-driven autonomy? It would seem so, although based on the available literature, subgoaling on unsatisfied preconditions does not *exemplify* what is colloquially meant by "goal-driven autonomy."

Third, we cannot yet precisely state what a GDA problem *is*. Originally, it was defined as a dynamic planning and execution problem over a partially-observable, non-deterministic, and dynamic system with continuous (i.e. non-discrete) effects, where *in addition* the agent can change its goals (Molineaux, Klenk, and Aha 2010). Today, state-of-the-art approaches to GDA only require the agent

[1]Adventure games do not meet the **offline planning** assumption, which requires integrated planning and execution.

be able to observe its environment for *discrepancies*, hypothesize *explanations* for them, and formulate goals to resolve the discrepancies that target the hypothesized explanations (Cox 2020; Dannenhauer, Muñoz-Avila, and Cox 2021). This suggests that even within GDA there are different *problem classes*. Without clarifying what the different problems *are*, a systematization is out of reach.

Due to the conflicted understanding of what problem GDA is intended to solve, several scholars have questioned the need to *distinguish* goal-driven autonomy in its own right, in favor of simply recasting GDA problems as specialized dynamic planning ones. For instance, Harland et al. (2014) have demonstrated that goal-driven autonomous behavior can be richly specified in terms of Belief-Desire-Intention (BDI) agent architectures. Is a commitment to models of BDI *necessary* to GDA? In contrast, Paredes and Ruml (2017) have argued that GDA is tantamount to dynamic planning across multiple levels of abstraction—they contend that this insight obviates the need for goal reasoning altogether. Does that preclude the utility of specifying certain problem classes *as* problems of goal-driven autonomy? *Is* there a class of GDA problems that *cannot* be systematically transformed into dynamic planning?

In view of this conflicted understanding and the questions that it poses, we propose it is worthwhile to re-examine the planning basis of goal-driven autonomy so that we may better understand its formal basis. In this paper, we do just that.

3 From Classical Planning to Classical Goal Reasoning: A Modest Proposal

Classical planning problems are well-defined: they contain a clear specification of the current situation, the desired end, and the available means to arrive at said end from the current situation (Newell, Shaw, and Simon 1960). These problems can be represented via STRIPS as follows.

Definition 1 (Classical Planning Problem) A tuple of the form $\mathbf{P} = \langle \mathcal{L}, I, g, \Lambda, f_{\text{cost}} \rangle$. $\mathcal{L}$ is a set of ground literals drawn from a finite and function-free first-order logical language, $I \subseteq \mathcal{L}$ is a closed-world initial state, and $g \subseteq \mathcal{L}$ is a set of goal conditions that must be satisfied. Λ is a set of action templates called *operators*. Each operator $\lambda \in \Lambda$ is a triple $\langle \text{PRE}(\lambda), \text{ADD}(\lambda), \text{DEL}(\lambda) \rangle$ of respective precondition, add, and delete lists, all subsets of $\mathcal{L}$. Each operator has an associated cost to execute, determined by the function $f_{\text{cost}} \colon \Lambda \to \mathbb{R}^{0+}$; unless stated otherwise, we use $f_{\text{cost}} = 1$.

The STRIPS representation is one way to encode a *state-transition system*, which specifies how some task environment can change as a result of applying *actions* in *states*.

Definition 2 (State-transition System) A quadruple of the form $\Sigma = \langle S, A, E, \gamma \rangle$. S is a set of states. A is a set of actions that represent state-transitions under the control of a planner. E is a set of events, or contingent state-transitions; these reflect the internal dynamics of the system *not* under the planner's control. $\gamma \colon S \times A \times E \to 2^S$ is a state-transition function that defines the *state space* of the system.

As noted by Ghallab, Nau, and Traverso (2004, hereafter, GNT), state-transition systems induce a directed graph that reflects the underlying (set-theoretic) *semantics* of planning, whereas STRIPS is a kind of *syntactic specification*. This distinction was introduced as part of GNT's *conceptual model for dynamic planning*, to indicate that (regardless of implementation) all dynamic planning variants perform operations that can be described in terms of Σ, making it a useful semantic basis of comparison.

The classical planning variant operates over a *restricted* form of the elements in Σ by adopting a set of 8 simplifying assumptions over the GNT model. These are:

A1 Finite Σ: $S \in \Sigma$ is finite.

A2 Static Σ: $E = \varnothing$.

A3 Fully Observable Σ: The agent has complete knowledge about the state of Σ.

A4 Deterministic Σ: $\forall s \in S, a \in A \colon |\gamma(s, a)| \leq 1$. Thus, γ maps onto S, instead of 2^S.

A5 Restricted Goals: Goals are either an explicit goal state $s_g \in S$ or a *family* of goal states $S_g \subseteq S$, defined as:

$$S_g = \{ s \mid (s \in S \in \Sigma) \wedge (s \models g \in \mathbf{P}) \} \tag{1}$$

The solution is *any* sequence of state-transitions that results in s_g or $s \in S_g$.

A6 Sequential Plans: A solution to a planning problem is a plan π, a linearly ordered finite sequence of actions; i.e. a *net* in A, written as $\pi \colon \mathbb{N} \to A$ or $\pi \colon A^{\mathbb{N}}$.

A7 Implicit Time: Actions A and events E have no duration; they are instantaneous state-transitions.

A8 Offline Planning: The planner is not concerned with any change that may occur in Σ *while* it is planning (it plans for the given initial and goal states).

Thus, the classical variant effectively disregards the controller and system in the full GNT model.

Given a problem $\mathbf{P}$ as in Definition 1, its underlying classical state-transition system $\Sigma(\mathbf{P})$ is defined as: $S \subseteq 2^{\mathcal{L}}$; A is composed of all $\lambda \in \Lambda$; $E = \varnothing$; and γ is piece-wise:

$$\gamma(s, a) = \begin{cases} s \setminus \text{DEL}(a) \cup \text{ADD}(a) & \text{if } \text{PRE}(a) \subseteq s \\ s & \text{otherwise} \end{cases}$$

The STRIPS classical variant compactly represents $\mathbf{P}$ as a *search problem* over the graph induced by $\Sigma(\mathbf{P})$ starting in $I \in S$ and ending in a state $s \in S_g \subseteq S$.

Further, STRIPS is a *specific* syntactic specification; i.e. it gives a *particular* form to classical planning operations, which can be conceptually described via a function φ as illustrated in Figure 2. In other words, a *generic* classical planner φ performs a mapping as follows:

$$\varphi \colon \overset{\text{state space}}{\gamma} \times \underset{\text{family of states } S_g \subseteq S}{S} \times \overset{\text{initial state } I \in S}{S} \rightarrow \underset{\text{plan}}{\pi} \tag{2}$$

Thus, φ takes the elements of a classical planning problem and maps them onto a plan structure. Of course, a specific *implementation* of φ (e.g. STRIPS) must ensure that π transforms the problem's initial state I to a state $s \in S_g \subseteq S$

that satisfies the goal, or fail by returning a null plan. But regardless of implementation, the *structure* of the mapping in Equation 2 is conceptually the same across the class of classical planners.

3.1 Classical Execution Problems

Our motivating example from §1 is not strictly a classical *planning* problem, as it relaxes assumptions A6 and A8. This re-introduces the task environment that was removed for classical planning, which in turn requires that we be more precise about the interplay between planner and system, as mediated by the controller. For brevity, we refer to what results from these relaxations as *classical execution problems*.

Definition 3 (Classical Execution Problem) A tuple of the form $\mathbf{X} = \langle \mathbf{P}, \mathbf{\Sigma} \rangle$, where $\mathbf{P}$ is a classical planning problem as in Definition 1, and $\mathbf{\Sigma}$ is a *classical task environment* defined as $\mathbf{\Sigma} = \Sigma(\mathbf{P})$.

Whereas the system $\mathbf{\Sigma}$ typically represents an *analog* task environment, Definition 3 admits it as a *discrete* one; i.e. the virtual world encompassing an adventure game. Conceptually, a classical execution problem $\mathbf{X}$ places one additional demand on solution plans: a plan π is a solution plan when its *execution trace* within $\mathbf{\Sigma} = \Sigma(\mathbf{P})$ yields a *system* state that satisfies the goal conditions. In order to precisely describe what this means, we must revisit the full GNT model.

Conceptual Gaps within the GNT Model Originally, the GNT model was presented as a theoretical device and was "...not meant to be directly operational" (Ghallab, Nau, and Traverso 2004, p. 9). Our rendition in Figure 2 is unique in that regard: it foregrounds ambiguities in the GNT model that lead to our motivating example's failure case.

Per the GNT model, the controller is responsible for *observing* the system $\mathbf{\Sigma}$ and using that information to drive the *dispatching* and *monitoring* of its given plan as it manifests inside the system. Because of the execution requirement, π is revised to be:

$$\underset{\text{the } \textit{current} \text{ plan}}{\pi} : \overset{\text{state we're in}}{S} \times \underset{\text{action sequence}}{A^{\mathbb{N}}} \to \overset{\text{next action to execute}}{A} \tag{3}$$

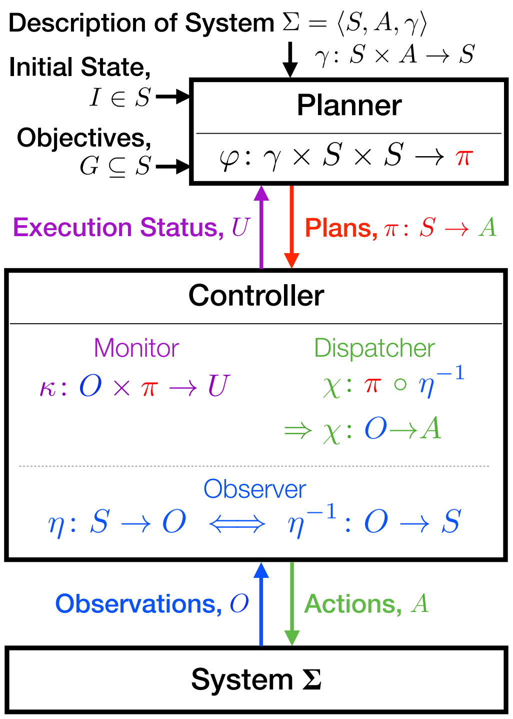
Further, the controller's components are defined as follows.

Observer This is the controller's capacity to perceive the system. It was originally defined by GNT as a function with corresponding inverse:

$$\eta\colon S \to O \tag{4}$$
$$\eta^{-1}\colon \underset{\text{observations } O \in \mathbf{\Sigma}}{O} \to S \tag{5}$$

Equation 4 relates states to *observations*. Generally, an observation $o \in O \in \mathbf{\Sigma}$ represents information from $\mathbf{\Sigma}$ obtained through (typically hardware-based) sensors.

In classical execution we make $O = S \in \mathbf{\Sigma}$ and η the identity function, which via its inverse in Equation 5 obtains the *current state* $s_{\text{current}} \in S \in \Sigma(\mathbf{P}) = \mathbf{\Sigma}$.



Description of System $\Sigma = \langle S, A, \gamma \rangle$
$\gamma\colon S \times A \to S$
Initial State, $I \in S$
Objectives, $G \subseteq S$
Planner
$\varphi\colon \gamma \times S \times S \to \pi$
Execution Status, U
Plans, $\pi\colon S \to A$
Controller
Monitor
$\kappa\colon O \times \pi \to U$
Dispatcher
$\chi\colon \pi \circ \eta^{-1}$
$\Rightarrow \chi\colon O \to A$
Observer
$\eta\colon S \to O \iff \eta^{-1}\colon O \to S$
Observations, O
Actions, A
System $\mathbf{\Sigma}$

Figure 2: Illustration of GNT's dynamic planning model with added details needed to define *classical execution*. We go beyond the GNT model by defining the pipeline of structural mappings needed to manifest a plan inside the system $\mathbf{\Sigma}$. This includes the newly added functions: (a) φ, representing a *generic* classical planner, (b) π, representing an *executable* classical plan, (c) χ, representing a *generic* system dispatcher, and (d) κ, representing a *generic* plan monitor.

Dispatcher This is the controller's capacity to find which action should be executed next according to both π and the current state of $\mathbf{\Sigma}$ per η^{-1}.

The dispatcher was only conceptually described by GNT. We formalize it as a composite *dispatch function* defined as:

$$\chi\colon \underset{\text{...within the current plan.}}{\pi} \overset{\text{...to find the action to do...}}{\circ} \overset{\text{Observe what state we are in...}}{\eta^{-1}} \tag{6}$$

Equation 6 effectively maps the observations to the plan's suggested next action. Stated directly:

$$\chi\colon O \to A \tag{7}$$

Of course, *classical executable plans* defined via Equation 3 make no use of state information (i.e. S is not in the domain of π), so effectively, the next action to do depends solely on the plan's action structure.

Monitor This is the controller's capacity to detect when plans fail. It was only conceptually described in the GNT model in terms of a *status code*, representing a report to the planner about the execution status of the current plan π.

A status code might prompt the planner to engage in *revision* and/or *re-planning* operations. But because this reporting process was not formalized, we discovered an interesting ambiguity: under GNT, plan revision and re-planning would only be necessary if we cannot find in π what to do next.

In classical execution, this might happen because we have achieved the goal or because the current state was not anticipated. Thus, in addition to looking within π, we also need information from the environment. This motivated our formulation of a *monitor function*, defined as:

$$\kappa\colon \underset{\text{Check if } \eta^{-1}(o) \models g \ \lrcorner}{\overset{}{O}} \times \pi \to \underset{\llcorner\ \text{status code}}{U} \tag{8}$$

3.2 Classical Execution Agents and Solutions

Conceptually, a solution plan to a classical execution problem must effect a change within the system $\mathbf{\Sigma}$ such that *when the system is observed by the agent* (via η^{-1}), the current state that is perceived $s_{\text{current}} \in S \in \Sigma(\mathbf{P}) = \mathbf{\Sigma}$ satisfies the goal; i.e. $s_{\text{current}} \in G$, where G is defined per Equation 1.

Because solution plans must be *constructively* assessed by the agent's perceptual abilities over $\mathbf{\Sigma}$, we reify the agent as:

Definition 4 (Classical Execution Agent) A quadruple of the form $\varrho = \langle \varphi, \eta, \chi, \kappa \rangle$, where φ is a planner function as in Equation 2; η is an observer function as in Equation 4; χ is a dispatcher function as in Equation 6; and κ is a monitor function as in Equation 8.

A classical execution agent ϱ has the capacity to effect an executable classical plan π within a classical task environment $\mathbf{\Sigma}$. We denote the execution trace of said plan as $trace(\pi, \varrho, \mathbf{\Sigma})$. An execution trace is the ordered sequence of alternating states and actions starting in the current state $s_{\text{current}} \in \mathbf{\Sigma}$. The sequence results from the application of action a_i obtained via the dispatcher $\chi \in \varrho$ to the state s_{i-1}; i.e. $trace(\pi, \varrho, \mathbf{\Sigma}) = [s_{\text{current}}, a_1, s_1, ..., a_m, s_m]$.

An executable classical plan π is a solution to a classical execution problem $\mathbf{X} = \langle \mathbf{P}, \mathbf{\Sigma} \rangle$ for agent ϱ iff its $trace(\pi, \varrho, \mathbf{\Sigma})$ results in a system $\mathbf{\Sigma}$ configuration that when perceived via the observer $\eta \in \varrho$ yields a state $s_{\text{current}} \in G$ as defined in Equation 1 from the goal conditions $g \in \mathbf{P}$.

3.3 Toward Classical Goal-driven Autonomy

Our goal-driven autonomy needs require reasoning about execution dynamics in what is arguably their simplest form: within computer-mediated virtual worlds. As mentioned, existing GDA solutions relax the A2 and A3 assumptions, which unduly complicates the algorithm needed to perform the reasoning we care about. As a comparison: it is as if all we needed was a classical planner and the only option we had available was a probabilistic one better-suited for Partially-observable Markov Decision Processes.

Instead of simplifying non-classical models, we opt to conservatively expand classical ones to "earn our complexity." To us, the elements within classical execution problems are necessary to justify the need for goal reasoning. At the same time, nothing within classical execution specifies what we actually need: a formal mapping that defines how the planner's input objectives G can systematically change.

When the goals can change, all potential changes implicitly define a goal *hyperspace* over Σ (Cox 2020): the states $S_{\mathbf{G}} \subseteq S \in \Sigma$ reachable from each other via transitions outside A (or $A \cup E$, as relevant). This makes the semantic specification of state-transition systems insufficient to fully describe *goal transitions*. To that end, we introduce a *goal-transition system* as an extension to Definition 2:

Definition 5 (Goal-transition System) A tuple of the form $\Theta = \langle \Sigma, G, R, \beta \rangle$. Σ is a state-transition system as in Definition 2, G is a set of goal states, R is a set of *goal refinements* that represent goal transitions, and $\beta\colon G \times R \times S \to 2^G$ is a goal-transition function.

The set of goal refinements R and transition function β are the goal reasoning analogue of state-transition system actions A and transition function γ.

Whereas STRIPS may be used to generate the classical form of the elements in Σ, we lack such a language to generate the elements in Θ. Notwithstanding, we define Θ's *classical* form as a restricted goal-transition system that adopts all of the classical execution assumptions and defines its elements as follows:

- $\Sigma = \langle S, A, \gamma \rangle$ is a classical state-transition system defined from $\mathbf{P} = \langle \mathcal{L}, I, g, \Lambda, f_{\text{cost}} \rangle$ as discussed earlier.
- $G \subseteq 2^S$, where $S \in \Sigma$. This reflects the A5 assumption: only families of goal states defined from sets of conditions $g \subseteq \mathcal{L}$ as in Equation 1 are admissible.
- R contains *goal refinements* r akin to actions $a \in A$, each of the form $r = \langle \text{PRE}(r), \text{ADD}(r), \text{DEL}(r) \rangle$. Instead of state transitions, they are *goal transitions*: they compactly describe a wide variety of dynamics that depend on PRE-adopted goals that might be *transformed* (possibly with PREconditions on their transformation), the newly ADDed goals that are *adopted*, and the DELeted goals that are *dropped*. Like actions, $\text{PRE}(r), \text{ADD}(r), \text{DEL}(r)$ are all subsets of $\mathcal{L}$. These dynamics manifest in a wide variety of modern goal reasoning systems (Roberts et al. 2014).
- β, like $\gamma \in \Sigma$, is defined piece-wise:

$$\beta(g, r, s) = \begin{cases} g \setminus \text{DEL}(r) \cup \text{ADD}(r) & \text{if } \text{PRE}(r) \in g \cup s \\ g & \text{otherwise} \end{cases}$$

The above definitions allow us to describe important classes of goal-transitions in the literature (cf. Cox, Dannenhauer, and Kondrakunta 2017), including:

Goal Formulation (Paisner et al. 2014): when an agent infers a new set of goal conditions from some state. This is a goal-transition of the form $\beta(\varnothing, r, s) = g$, where the refinement $r = \langle \varnothing, g, \varnothing \rangle$.

Goal Change (Choi 2011): when an agent transforms an existing goal into another. This is a goal-transition of the form $\beta(g, r, s) = g'$, where the refinement $r = \langle g, g', g \rangle$.

4 Classical Goal-driven Autonomy

In the original definition of goal-driven autonomy, Molineaux, Klenk, and Aha (2010) relaxed assumptions A2, A3, A4, A6, A7, and A8. They further relaxed a tacit assumption not codified in the original 8, which we make explicit:

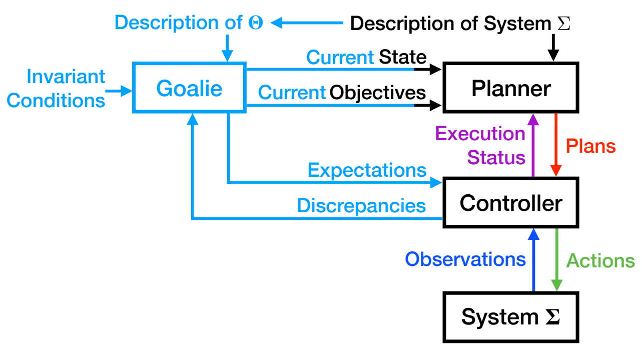


Figure 3: Illustration of our conceptual model for goal-driven autonomy. To us, goal-driven autonomy requires augmenting the dynamic planning model with a **goalie**, responsible for goal reasoning on as we detail in §4.1.

A9 Static Goals: The planner's goal does not change.

While this assumption may seem trite by now, we believe its relaxation is what conceptually distinguishes the goal-driven autonomy problem class from automated planning.

It is worth noting that Θ is not a classical goal-driven autonomy problem *per se*, much like Σ is not a classical planning problem. That is, Σ defines the *planning problem space* wherein a planner searches for a solution. By analogy, Θ defines the *goal-driven autonomy problem space* wherein a goal-reasoning system—what we call a **goalie** in our conceptual model (Figure 3)—searches for a solution.

4.1 Goal-driven Autonomy Problem Definition

The literature on potential reasons that might prompt a goalie to act is vast (Cox 2007; Weber, Mateas, and Jhala 2010; Muñoz Avila et al. 2010; Talamadupula et al. 2010; Laird, Rosenbloom, and Newell 2012). The clearest articulation of a common underlying feature is via work by Vattam et al. (2013) and Cox (2013), who independently argued that goal reasoning dynamics manifest as a consequence of an agent's (domain-specific) *anomaly detection mechanisms*.

We propose an alternate, albeit consistent, view: that the situations that would *trigger* those anomaly detection mechanisms have enough commonality to support their specification in a domain-independent way. We posit that the common ground across goalies is their drive to fix broken *invariant conditions* $\mathcal{I}$—conditions that collectively define a state of equilibrium the agent must maintain.[2] If all invariant conditions are maintained, no goal reasoning is necessary. On the other hand, *any* invariant condition being broken is *sufficient* to check if goal reasoning is required.

[2]In this light, the anomaly detection mechanisms implemented in goalies to date are implicitly describing *specific* invariant conditions to be maintained. If this is so, we should be able to formalize these tacit invariants in terms of execution problems and solutions, goal-transition systems, and formal properties thereof. While space limitations preclude our formal development of these ideas, our future work aims to do so.

Conceptually, in order to check against invariant conditions in $\mathcal{I}$ being violated during execution, the goalie[3] must sense what the current state is, which is already carried out by the controller. This suggests to us a natural division of responsibilities: the goalie computes the system $\mathbf{\Sigma}$ *expectations* to monitor for based on $\mathcal{I}$ and an input description of Θ, whereas the controller computes the difference between the goalie's expectations and the observed system $\mathbf{\Sigma}$ state. We term these differences *discrepancies* and posit that they prompt the goalie to engage in goal reasoning as appropriate. While we cannot formalize all these operations due to space, they are consistent with several goalies to date (cf. Molineaux, Klenk, and Aha 2010; Roberts et al. 2014; Dannenhauer, Muñoz-Avila, and Cox 2021).

To perform the goal reasoning itself, goalies need access to a collection of "goal operators"—akin to the action operators Λ in classical planning—to support flexible goal reasoning dynamics. We call these template goal operators *modifications*, defined as generic forms of the refinements $R \in \Theta$.

We can now define the *goal-driven autonomy problem* that represents the goalie's input description of Θ, and which can describe our motivating adventure game context.

Definition 6 (Classical Goal-driven Autonomy Problem) A tuple of the form $\mathbf{G} = \langle \mathbf{X}, \mathcal{I}, M \rangle$, where $\mathbf{X} = \langle \mathbf{P}, \mathbf{\Sigma} \rangle$ is a classical execution problem as in Definition 3, $\mathcal{I} \subseteq S \subseteq \mathbf{\Sigma}$ is a set of invariant conditions that define states to maintain in the system, and $\mathbf{M}$ is a set of template refinements called *modifications*—each defined as a triple $m = \langle \text{PRE}(m), \text{ADD}(m), \text{DEL}(m) \rangle$ of respective precondition, add, and delete lists, all subsets of $\mathcal{L} \in \mathbf{P}$.

Notably, Definition 6 tracks one goal formula: $g \in \mathbf{P}$. Refinements in M manifest as changes to this goal.

4.2 Toward Goalie-independent Problem Specs

Our proposal to encode invariants as the common feature to goal-driven autonomy paves the way toward a goalie-independent problem specification language akin to STRIPS, which itself affords planner-independent specification of closed problems. To that end, we introduce GRUPS: the Goal Reasoner ∪ Problem Solver, a speculative STRIPS-like language that aims to afford the expression of classical problems as per Definition 6. Ultimately, this means one may eventually use GRUPS to arrive at a precise origin of the elements in Θ from Definition 5 much like STRIPS was used to generate those of Σ from Definition 2. Here, we limit ourselves to illustrating how our motivating example might be encoded via GRUPS.

A Modest Extension to PDDL GRUPS may be feasible to develop via four extensions to PDDL 3 (Gerevini and Long 2005). We review each to explain their intended purpose.

In PDDL 3, *constraints* are "over possible actions in the plan and intermediate states reached by the plan." Constraints operate on both binary and numerical values, and

[3]We discuss the goalie *as if* it were a stand-alone component with a single responsibility (*i.e.* goal reasoning) working alongside a planner and controller. While this helps us decouple the scope of goalie *v.* planner *v.* controller concerns, we do not intend to prescribe the choice of how to implement this architecture in practice.

```
(:constraints ...) ; as per PDDL 3
(:goals ...) ; problem-independent goals go here
(:action pick-up
  :parameters (?pc - user ?i - item ?l - loc)
  :precondition (and (at ?pc ?loc) (at ?i ?loc))
  :effect (and (not (at ?i ?loc)) (has ?pc ?i)))

(:action give
  :parameters (?g - char ?r - char ?i - item ?l - loc)
  :precondition (and (at ?g ?l) (at ?r ?l) (has ?r ?i))
  :effect (and (not (has ?r ?i)) (has ?g ?i)))

(:refinement fetch-quest
  :parameters (?p - char ?np - char ?i - item ?l - loc)
  :precondition (and (at ?pc ?l) (at ?npc ?l))
  :effect (and (has ?npc ?i)))
```

Listing 1: Sample GRUPS goal refinements in PDDL-like syntax. The *fetch-quest* refinement affords an adventure game AI to adopt a goal from a non-player character. This goal would prompt planning toward obtaining the character-requested item and giving the item to said character.

```
(:constraints ...); problem-specific invariants
(:goals ; implicitly disjuncted goals:
  (:goal (...)) ; g1 or
  (:goal (and ((has player mcguffin)))) ; g2 or
  (and ; the conjunction of goals g3
    (:goal (...))
    (:goal (...))))
```

Listing 2: Sample GRUPS problem file in PDDL-like syntax. Above, the *goals* keyword affords specifying the goals managed by a GDA agent, as an implicitly disjuncted set of (possibly unary) goal sets. The goal g2 appears as a result of the *fetch-quest* refinement in Listing 1.

manifest via several built-in predicates; e.g. *always*, *sometime*, *within*, *at-most-once*, and several other built-in predicates. Where GRUPS might need to differ from PDDL 3 is for allowing constraints to be defined per-problem as well. Problems can define how they must be solved, extending restrictions given to the goalie from the domain. Constraints would then be able to represent *invariants* as described earlier, and would not require adding a new construct.

Refinements are a GRUPS-extension to the PDDL 3 spec, as demonstrated in Listing 1. Refinements operate on a goal in ways similar to how actions modify the system state. They are syntactically specified similar to PDDL actions.

Goals represent the biggest syntactic departure between PDDL 3 and GRUPS. We define *goals* as a set of *goal*-s; the latter defined per the original PDDL3 spec as demonstrated in Listing 2. Our sketch here supports managing different goals at a time via one goal formula, and we feel it is sufficiently human-readable to support problem authorship.

5 Discussion

Dynamic planning problems are semantically *closed* to new goals because by definition the problem goals are *presumed* (Ghallab, Nau, and Traverso 2004). In contrast, goal-driven autonomy problems are *characterized* in terms of goals that must be refined and by definition are *open* to them. Dynamic planning thus targets *closed problems*, and represent the challenge of articulating *how* goals are solved, akin to the human cognitive process of *problem solving* (Newell, Shaw, and Simon 1960). In turn, goal-driven autonomy targets *open problems*, and represent the challenge of articulating *which* goals need solving, akin to the human cognitive process of *problem recognition* (Cox 2013).

Notably, we have not yet defined the algorithms that would drive a classical goal-driven agent. While space prevents us from doing so, we briefly sketch the functional mappings that might be added to Definition 3 to support solving classical GDA problems. Let ω be a *generic* goalie function akin to its planner counterpart from Equation 2. Its output is three-fold: (1) an initial state based on the system's current one with (2) goals for the planner to pursue, and (3) expectations our controller will be tasked with monitoring.

This then requires a *generic discrepancies detector* function, which we denote as Ω. Together with κ, the overall architecture supports defining systematic ways to decide on whether *plan shifts* are necessary or if *goal shifts* are. We conclude our theory sketch with a preliminary formal definition of a goal-driven agent: a tuple, $\Psi = \langle \varphi, \eta, \chi, \kappa, \omega, \Omega \rangle$, with elements as defined throughout.

6 Conclusion

With this combined theoretical foundation, we discovered several things. First, we articulated a novel and parsimonious problem-based distinction between automated planning and goal reasoning (Definition 3 *v.* Definition 6). Second, we illustrated how our formulation can potentially reconcile a wide variety of features that presently motivate the wide array of approaches to goal-driven autonomy (§3). Third, we've identified that core assumptions around GDA are not strictly necessary to define interesting goal reasoning problems; relatedly, goal reasoning is necessary upon relaxing a tacit assumption we made explicit: the static goals (A9) assumption (§4). Fourth, we defined a novel semantic foundation for GDA via Definition 5. Fifth, in §4 we sketched a new goalie-agnostic language—GRUPS—that we will continue to expand in future work. Finally, we demonstrated a potentially interesting class of GDA problems: those involved in developing an AI capable of playing adventure games. We hope others are encouraged to systematically explore this problem class with us.

Acknowledgments

This material is based upon work supported by the Office of Naval Research, Naval Research Laboratory under Grant No. N00173-21-Q-0138, and supported by the National Science Foundation under Grant No. #2046294. We also wish to thank the anonymous reviewers who were tremendously helpful with their comments during peer review. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the supporting agencies.

References

Aha, D. W. 2018. Goal Reasoning: Foundations, Emerging applications, and Prospects. *AI Magazine* 39(2): 3–24.

Black, M. L. 2012. Narrative and Spatial Form in Digital Media: A Platform Study of the SCUMM Engine and Ron Gilbert's The Secret of Monkey Island. *Games and Culture* 7(3): 209–237.

Cardona-Rivera, R. E.; Zagal, J. P.; and Debus, M. S. 2020. GFI: A Formal Approach to Narrative Design and Game Research. In *Proceedings of the 13th International Conference on Interactive Digital Storytelling*, 133–148.

Choi, D. 2011. Reactive goal management in a cognitive architecture. *Cognitive Systems Research* 12(3-4): 293–308.

Cox, M.; Dannenhauer, D.; and Kondrakunta, S. 2017. Goal operations for cognitive systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

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

Cox, M. T. 2013. Goal-driven autonomy and question-based problem recognition. In *Proceedings of the 2nd Annual Conference on Advances in Cognitive Systems*, 29–45.

Cox, M. T. 2020. The Problem with Problems. In *Proceedings of the 9th Annual Conference on Advances in Cognitive Systems*.

Dannenhauer, D.; Muñoz-Avila, H.; and Cox, M. T. 2021. Expectations for Agents with Goal-driven Autonomy. *Journal of Experimental & Theoretical Artificial Intelligence* 33(5): 867–889.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2(3-4): 189–208.

Gerevini, A.; and Long, D. 2005. Plan Constraints and Preferences in PDDL3. Technical report, Department of Electronics for Automation, University of Brescia.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. Elsevier.

Harland, J.; Morley, D. N.; Thangarajah, J.; and Yorke-Smith, N. 2014. An Operational Semantics for the Goal Life-Cycle in BDI Agents. *Autonomous agents and multi-agent systems* 28(4): 682–719.

Hawes, N. 2011. A survey of motivation frameworks for intelligent systems. *Artificial Intelligence* 175(5): 1020–1036. ISSN 0004-3702. doi:https://doi.org/10.1016/j.artint.2011.02.002. Special Review Issue.

Kambhampati, S.; Knoblock, C. A.; and Yang, Q. 1995. Planning as refinement search: a unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence* 76(1): 167–238.

Laird, J.; Rosenbloom, P.; and Newell, A. 2012. *Universal subgoaling and chunking: The Automatic Generation and Learning of Goal Hierarchies*, volume 11. Springer Science & Business Media.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. E. 1998. PDDL–The Planning Domain Definition Language. Technical report, Yale Center for Computational Vision and Control, New Haven, CT, USA.

Molineaux, M.; Klenk, M.; and Aha, D. 2010. Goal-driven Autonomy in a Navy Strategy Simulation. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*.

Muñoz Avila, H.; Jaidee, U.; Aha, D. W.; and Carter, E. 2010. Goal-driven autonomy with case-based reasoning. In *International Conference on Case-Based Reasoning*, 228–241. Springer.

Newell, A.; Shaw, J. C.; and Simon, H. A. 1960. Report on a general problem-solving program for a computer. In *Proceedings of the International Conference on Information Processing*, 256–264.

Paisner, M.; Cox, M.; Maynord, M.; and Perlis, D. 2014. Goal-driven autonomy for cognitive systems. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 36.

Paredes, A.; and Ruml, W. 2017. Goal Reasoning as Multilevel Planning. In *Proceedings of the ICAPS-17 Workshop on Integrated Execution of Planning and Acting*.

Riedl, M. O.; and Bulitko, V. 2013. Interactive Narrative: An Intelligent Systems Approach. *AI Magazine* 34(1): 67–77.

Roberts, M.; Vattam, S.; Alford, R.; Auslander, B.; Karneeb, J.; Molineaux, M.; Apker, T.; Wilson, M.; McMahon, J.; and Aha, D. W. 2014. Iterative goal refinement for robotics. In *Proceedings of the 2014 ICAPS Workshop on Planning and Robotics*.

Talamadupula, K.; Benton, J.; Kambhampati, S.; Schermerhorn, P.; and Scheutz, M. 2010. Planning for Human-robot Teaming in Open Worlds. *ACM Transactions on Intelligent Systems and Technology* 1(2): 1–24.

Vattam, S.; Klenk, M.; Molineaux, M.; and Aha, D. W. 2013. Breadth of Approaches to Goal Reasoning: A Research Survey. In *Goal Reasoning: Papers from the ACS Workshop*, 111–128.

Weber, B. G.; Mateas, M.; and Jhala, A. 2010. Case-based Goal Formulation. In *Proceedings of the AAAI Workshop on Goal-Driven Autonomy*.

Weld, D. S. 1994. An Introduction to Least Commitment Planning. *AI Magazine* 15(4): 27.