Proceedings of the Ninth Annual Conference on Advances in Cognitive Systems Submitted 9/2021; published 11/2021

Deep Goal Reasoning: An Analysis

Weihang Yuan WEY218@LEHIGH.EDU
Hector Munoz-Avila HEM4@LEHIGH.EDU

Abstract

We analyze the goal reasoning capabilities of two-layer hierarchical reinforcement learning systems based on the architecture of Kulkarni et al. (2016)'s h-DQN. In our analysis we position these architectures in the broader context of goal reasoning: mapping their capabilities relative to the OODA framework and Aha (2018)'s goal dimensions. We then model their goal reasoning capabilities using formal grammars. We analyze the expressiveness of state trajectories of two existing variants and illustrate our results with an empirical evaluation.

1. Introduction

The ability of an agent to change its goals in response to observations of an environment is a hallmark of cognitive architectures. Examples include ICARUS that learns a goal hierarchy (Choi & Langley, 2018), SOAR that generates new goals when encountering impasses (Laird, 2019), and MIDCA's perception-action cycle, which formulates goals at both the object and meta level (Cox et al., 2016). More generally, goal reasoning has been a recurrent research topic in cognitive systems. For example, goal-driven autonomy aims to control the focus of an agent's planning activities by formulating goals dynamically when resolving unexpected discrepancies in the world state (Klenk et al., 2013).

Given the recurrent interest in goal reasoning systems, it is not surprising that deep learning architectures that reason with goals explicitly have been proposed. For example, h-DQN (Kulkarni et al., 2016) combines hierarchical deep reinforcement learning (RL) and goal reasoning. It is a two-level hierarchical RL system: a Goal Selector at the top level generates goals and an Actor at the lower level takes actions to achieve those goals. Since its introduction, several variants of this, what we will call GoalAct architectures in this paper, have been proposed (Vezhnevets et al., 2017; Mousavi et al., 2019); these variants differ on what kinds of inputs are given to the Goal Selector and the Actor and when the Actor cedes back control to the Goal Selector.[1]

In order for these kinds of deep learning goal reasoning architectures are to be adopted by cognitive systems we need to have a better understanding of their capabilities and limitations. Specifically, in this paper we will address the following two questions:

- How are these GoalAct architectures situated relative to the goal reasoning paradigm?
- What kinds of outputs are these GoalAct architectures capable of generating?

1. In h-DQN the two components are referred to as the meta-controller and the controller; we adopted Goal Selector and Actor to emphasize their function.

In order to answer these two questions, we connect three ideas: GoalAct architectures, goal reasoning, and formal grammars. Goal reasoning is the study of agents that are capable of reasoning about their goals and changing them when the need arises (Aha, 2018). Goal reasoning agents follow the OODA loop (Boyd, 1995): they observe the environment, orient to filter the information needed to select the goals, decide on which goal to pursue, and act to achieve that goal. Formal grammars are collections of rules $\alpha \rightarrow \beta$, each indicating how to transform a string (i.e., a sequence of characters) into another string (Sipser, 2012). Aside from its inferencing capabilities (i.e., transforming an input string by repeatedly applying rules) (Ambite & Knoblock, 2001), formal grammars facilitate expressiveness analysis; that is, conclusively stating whether particular strings transformations are possible or not (Erol et al., 1994).

2. Background

Formal Grammars. Unrestricted formal grammars consist of rules that are used to transform strings. Rules have the form: $\alpha \rightarrow \beta$, where α and β are strings concatenating variables and constants. The string α must have at least one variable. Variables denotes multiple possible strings consisting of constants only. Strings with constants only are terminal and cannot be further transformed. We will provide detailed examples later, but briefly, the transformation procedure starts with a single variable, called the start variable, then the rules are repeatedly applied until a string consisting solely of constants is generated.

Reinforcement Learning. In this work, we consider episodic tasks. That is, the process eventually terminates when reaching the terminal state τ (although it can be restarted an arbitrary number of times). The environment consists of a finite set of states $\mathcal{S}$, a finite set of actions $\mathcal{A}$, a transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$, and a reward function $\mathcal{R} : \tilde{\mathcal{S}}^{\leq k} \mapsto \mathbb{R}$, where ($\tilde{\mathcal{S}}^{\leq k}$ is the set of all possible concatenation of, up to k, states in $\mathcal{S}$. and the number k is fixed.) At each time step t, the agent observes a state $s_{[t]} \in \mathcal{S}$ and takes an action $a_{[t]} \in \mathcal{A}$. At the next time step, the environment transitions to $s_{[t+1]}$ and the agent receives a reward $r_{[t+1]}$.[2] The process repeats until the terminal state is reached. The objective of the agent is to take actions in a way that maximizes the cumulative future reward: $G_t = \sum_{i=t}^{T} \gamma^{i-t} r_{[i+1]}$, where γ is the discount rate, $0 \leq \gamma \leq 1$, and T is the number of steps in the episode. A solution and approximations of a solution are represented as a stochastic policy: $\pi : S \times A \mapsto [0, 1]$, indicating the probability $\pi(s, a)$ of choosing action a in state s. A special case is when the policy is deterministic: for every state s, there is one action a, for which $\pi(s, a) = 1$. For any other action a', $\pi(s, a') = 0$.

3. GoalAct Architecture

In this paper we refer to hierarchical reinforcement learning agents (see Figure 1) that follow the design principles of h-DQN's 2-layer architecture as GoalAct architectures. The goal selector receives as input the state s possibly augmented with additional inputs and generates the next goal g to be achieved. The Actor takes actions based on g, the state s and possibly other inputs. In h-DQN, for

2. A subscript with brackets denotes a time step. For example, $s_{[0]}$ is the environment's state at time step 0. Without the brackets, the subscript denotes a distinct element in a set.

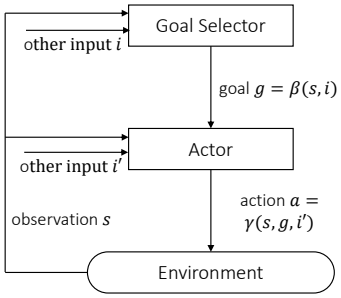


Figure 1: The GoalAct architecture. The Goal Selector chooses a goal g as a function β of the state s and information i. The Actor selects an action as a function γ on g, s and information i'.

example, the Goal Selector receives the state s and outputs the goal g whereas at the bottom-level it receives as input s and g and outputs an action a. The action a is executed resulting in a new state s'. The Goal Selector is learning which goal should be selected to maximize the rewards from the environment. The Actor is learning how best to achieve the given goal g. To accomplish this, the Actor receives a binary reward of 1 if the goal is achieved and 0 if not.

Instances of this architecture vary on what input is given at each level and when the Actor cedes back control to the Goal Selector:

- **Inputs.** Examples of inputs include: (i) in h-DQN the state is a screenshot of the Atari game. It has no other inputs (i.e., $i = i' = \emptyset$); (ii) in Mousavi et al. (2019), the state is an image of the part of the the map that has been discovered. Additional inputs, $i = i' =$ *"a representation of the map that has not been explored and locations visited previously by the agent"*. (iii) In the FuNs system $i = i' =$ *"the list of states visited previously."*

- **Given control back to Goal Selector.** Examples of when the Actor gives back control to the Goal Selector includes: (i) In h-DQN, the Actor continues generating actions until a state s_g is reached where g is satisfied. (ii) In Mousavi et al. (2019), where the agent is performing exploration tasks in an unknown map, goals are directional vectors indicating the general direction the agent should explore, the Actor generates a fixed number of actions moving the agent in the given direction and then control is given back to the Goal Selector. (iii) In FuNs (Vezhnevets et al., 2017), the Actor also executes a fixed number of actions.

The goal selector and the actor operate at different temporal scales: at the start of the process, the initial state $s_{[0]}$, possibly augmented with additional information, is given as input to the Goal Selector. The Goal Selector selects a goal g. This goal, state $s_{[0]}$, and additional information is given as input to the Actor, which selects an action $a_{[1]}$ and applies it, resulting in a state $s_{[1]}$. The process repeats itself; the Actor then selects an action $a_{[2]}$ based on state $s_{[2]}$, g and any additional information. Eventually, at some state $s_{[n]}$ the Actor cedes back to the Goal Selector and the cycle starts again.

4. Goal Reasoning with GoalAct

Cox (2017) formalizes the goal reasoning process as a goal transformation rule: $\beta(s, g) \rightarrow g'$ where s is the current state, g is the current goal and g' is the next goal.

In general, the goal-plan architecture is amenable to implementing a goal transformation function: $\beta(s, i) \rightarrow g'$ where i is some additional information such as the current goal or the prior states visited.

Using this formalization, we see that h-DQN implements a simplification of this transformation (we call this the basic goal transformation): $\beta(s, \_) \rightarrow g'$ where the current state is used to select the next goal but not the current goal.

The GoalAct architecture is amenable to extensions: as previously discussed, Mousavi et al. (2019) and FuNs takes as input i = *"list of states visited"*, $\tilde{s}$. This memory-based architecture is defining a goal transformation function (we call this the mnemonic goal transformation): $\beta(s, \tilde{s}) \rightarrow g$.

We now analyze the GoalAct architecture relative to the observe, orient, decide, act (OODA) loop as described in Aha (2018):

- **Observe.** Observe refers to raw sensor readings from the environment and are represented in the form of a state s, possibly partially observed. That is, s is a representation of what the agent sees but not necessarily captures the whole environment on which the agent is operating. This maps the GoalAct architecture, where the state can be an image or a vector representing a map annotated with the current location of the agent.

- **Orient.** The orient steps focuses the agent's attention. In goal-driven autonomy, for example, the orient step is implemented in several steps: the agent generates expectations X of the outcome of its actions (e.g., X could be expected state). These expectations are then matched against the observed state, $o(s)$ (e.g., checking if $X = o(S)$). If a discrepancy occurs (e.g., $X \neq o(S)$), an explanation e is generated. This explanation is used to generate a goal. The orient step is completely bypassed in existing implementations of GoalAct. In GoalAct, goals are generated only when the Goal Selector is in control. This means any implementation of orient will require the ability to prevent the Actor from generating the next action. Current implementations stop the Actor when: (i) a state s_g is reached where g is satisfied or (ii) after executing a fixed number of steps. They key point is that the decision for the Actor to continue is implicit and not part of the internal network representation in the Actor. The decision to stop or continue is based on the the observed state s and other inputs i'.

- **Decide.** Decide involves selecting which goals to manage and among those managed which goals to pursue. In the architecture this is simplified: there is only one goal that is pursued at any point of time. This goal is selected among the list of all possible goals, which remains fixed for the lifetime of the system, not just within one episode. As we will observe when we discuss implementation details, the reason for this constraint is that goals are represented by an output vector in the deep learning architecture implementing the Goal Selector. While this list can be very large, it remains fixed. In principle, any changes in the list will require to re-learn by running the system on all episodes experienced so far.

- **Act.** Once a goal has been decided, the control of the agent is given to the Actor, which takes actions in order to achieve the goal. The effects of each action causes the environment to transition to from one state to the next. The Actor continues to take actions until either the goal is achieved or the some terminal condition is met.

Aha (2018) presents a taxonomy of goals classes. We now examine how the GoalAct architecture fits within that taxonomy.

- **Declarative or Procedural goals.** There are examples for both kinds of goals. h-DQN uses declarative goals, that is, they refer to desired state the agent wants to be in (e.g., in the Montezuma's Revenge game, one of the goals is for the agent to reach a location with a key). In Mousavi et al. (2019), goals are procedural. That is the goals indicate actions to take. In this case the goals indicate the general direction the agent should move towards.

- **Concrete or abstract goals.** h-DQN have a concrete goal, such as reaching a particular location. In Mousavi et al. (2019) and FuNs, the goals are general directions for the agent to navigate toward.

- **Durative or static-time goals.** Goals are static. We know of no mechanism to reason with durative goals in this architecture.

- **Knowledge goals or regular goals.** Knowledge goals refer to goals that generate new knowledge. For instance, information gathering systems would generate a goal to query a database so they can obtain information (e.g., a map of a city) to continue planning (e.g., reach a location within that city). We don't know of any instance of GoalAct using knowledge goals. The architecture would allow them insofar they are known in advance to be plausible goals.

- **Conditional goals.** Conditional goals are goals that can only be triggered when certain conditions are met. While we know of no implementation using conditional goals, we also don't see any limitations in GoalAct that would preclude such goals since the purpose of the Goal Selector is to learn under which state conditions (and other input i), a goal should be selected.

- **Goals can be interrupted.** While the agent is trying to achieve a goal, circumstances in the environment may change, making it undesirable to continue pursuing that goal. We know of no implementations of the GoalAct architecture with interruptible goals capability; that is, the Actor will stop pursing goal g and cedes back control to the Goal Selector to generate a new goal when some conditions are observed. But there is nothing in the architecture that would prevent it from interrupting a goal. In fact, as discussed before Mousavi et al. (2019) arbitrarily interrupts execution of a goal after performing a fixed number of actions.

The final point we want to discuss is the issue of representation of the goals. In Goal Reasoning agents, goals are represented using first-order logic accounting for the underlying mechanisms such as AI planning, used to reason about the goals. In the GoalAct architecture, goals are represented as a vector, where each entry has the current estimated value $v(g)$ for each goal g. These values are updated using deep reinforcement learning techniques where the weights θ of the network are

updated, and the goals are selected based on the weights θ and the policy π encoded in the network. So for instance in h-DQN, the expression $g \sim \pi(.|s;\theta)$ is used (Sutton & Barto, 2018). It indicates that g is selected as a function of π, s, and θ. During learning, when the agent is balancing exploration and exploitation, it may use the so-called ϵ-greedy strategy; namely, the agent will, with some frequency, $1-\epsilon$ selects the goal g with the highest value $v(g)$ (exploitation) and with some frequency, ϵ, a random goal. As the agent convergences, ϵ tends towards zero. In our analysis in the next section, we assume learning has occurred and therefore $\epsilon = 0$ (i.e., the agent pick the goal with the highest value).[3]

5. Expressiveness Analysis

We now turn our attention to provide insights into the kinds of state sequences that the GoalAct architecture can and cannot generate. We examine two variants of the GoalAct architecture, one with memory and one without memory, to determine the kinds of state sequences each is able to generate. The key insight is to model the following generalization of Cox's goal transformation rule:

$$\beta(s, i) \rightarrow g$$

We examine two cases: (1) when there is no additional input besides the state (i is empty) and (2) when the input is the history of previous visited states ($i = \tilde{s}$). We formulate the following grammar rule:

$$i\ s\ \langle \text{GOAL} \rangle \rightarrow i\ s\ g\ \langle \text{ACT} \rangle,$$

where $\langle$GOAL$\rangle$ is a variable representing the Goal Selector and $\langle$ACT$\rangle$ is a variable representing the actor.

We begin with an analysis of the basic architecture, as implemented in h-DQN, where the Goal Selector defines a goal transformation function $\beta(s, \_) \rightarrow g$. That is the next goal is selected as a function of the current state s. Consider a one-dimensional corridor environment shown in Figure 2 (top), which is adapted from Kulkarni et al. (2016). The corridor consists of 7 states from left to right: $s_0, \ldots, s_6$. The agent starts in s_3 (circle) and can move left or right. s_0 is the terminal state. In the "one-visit" task, the agent receives a reward of $+1$ if it visits s_6 (asterisk) at least once and then reaches s_0; otherwise, the reward is $+0.01$. Figure 2 (a) shows a trajectory that produces a reward of $+1$. A GoalAct architecture agent implementing the goal transformation function $\beta(s, \_) \rightarrow g$ is able to generate this trajectory and obtain the maximum reward. Now consider the "two-visits" task: the agent must visit s_6 at least twice to receive a reward of $+1$. We will show that it is impossible for these architectures, including h-DQN, to deterministically generate a trajectory (e.g., Figure 2 (b)) that solves "two-visits" task.

In our analysis we compare two instances of the GoalAct architecture:

- when the Goal Selector implements the goal transformation function $\beta(s, \_) \rightarrow g$, we call this *Basic GoalAct*.

3. In our experiments, during learning, we use a softmax distribution to select goals, which serves the same purpose of balancing exploration and exploitation as ϵ-greedy.

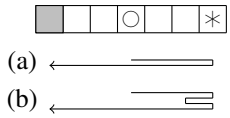


Figure 2: The corridor environment (top). The starting state is circled. The terminal state is gray. In the 2-visit corridor task, trajectory (a) results in a reward of $+0.01$; trajectory (b) results in a reward of $+1$.

- when the Goal Selector implements the goal transformation function $\beta(s, \tilde{s}) \rightarrow g$, where $\tilde{s}$ are the sequence of all states previously visited. We call this *Mnemonic GoalAct*.

We refer to GoalAct to describe either of these variants. When GoalAct interacts with an environment, the Goal Selector receives states and generates goals, and the Actor takes actions, forming a trajectory of states. For example, the string $s_3 s_6 s_0$ represents the state trajectory in Figure 2 (a). To analyze the expressiveness of GoalAct, we define two special types of grammars that generate strings of trajectories. A synopsis of our arguments is as follows:

1. We describe how GoalAct generate goals in an environment. Then we define two types of grammars: *basic* and k*-mnemonic*, to represent Basic GoalAct's and Mnemonic GoalAct's goal generation respectively.

2. We prove that a basic grammar is a special case of a k-mnemonic grammar. In other words, for any basic grammar, there exists a k-mnemonic grammar such that both grammars generate the same strings.

3. We prove that no basic grammar can generate a particular string. However, there exists a k-mnemonic grammars that can generate this string.

Numeral 2 implies that any state sequence generated by Basic GoalAct, can also be generated by Mnemonic GoalAct. Numeral 3 implies that that there are state sequences that Mnemonic GoalAct can generate that are impossible to generate with Basic GoalAct. The arguments combined imply that Mnemonic GoalAct is strictly more expressive than Basic GoalAct. Moreover, our analysis provides insights into the kinds of state sequences each can generate.

In our analysis, we make the assumption of deterministic policies: the system selects goals and actions deterministically. RL systems oftentimes use an exploratory stochastic policy. For example, h-DQN uses an ϵ-greedy policy to select goals and actions during training. As learning proceeds, ϵ is gradually reduced to zero. We want to analyze the system's behavior when $\epsilon = 0$ and hence make this assumption.

5.1 Expressive Power of Basic GoalAct

Based on the operating cycle of Basic GoalAct, we construct the following production rules to capture its expressive power:

(a) There can be one or more starting states. For every starting state s, we add one rule of the form: $S \rightarrow s\langle\text{GOAL}\rangle$. S is the starting variable and $\langle\text{GOAL}\rangle$ is a variable representing the Goal Selector. This ensures that the first symbol appearing in the resulting state trajectory string is an starting state and control is then given to the Goal Selector.

(b) For every state-goal assignment of the goal transformation function $\beta(s, \_) \rightarrow g$, we add one rule of the form: $s\langle\text{GOAL}\rangle \rightarrow sg\langle\text{ACT}\rangle$. $\langle\text{ACT}\rangle$ is a variable representing the Actor. These rules can be read as follows: if the agent is in state s and the Goal Selector is in control, then goal g is generated and control is given to the Actor.

(c) From t to $t+n$, the Actor receives as input a *(state,goal)* pair $(s_{[t+i]}, g_{[t]})$ and takes an action $a_{[t+i]}$, where $0 \leq i \leq n-1$. The iteration has three possible outcomes:

   i. The Actor returns control to the Goal Selector when $g_{[t]}$ is achieved in $s_{[t+n]}$ and $s_{[t+n]}$ is not the terminal state. This is described by the rule $sg\langle\text{ACT}\rangle \rightarrow ss'\langle\text{GOAL}\rangle$, where $s = s_{[t]}$ is the state when the Goal Selector gave control to the Actor, $g = g_{[t]}$ is the goal selected by the Goal Selector, and $s' = s_{[t+n]}$ is a state satisfying g. The state s' is added to the end of the string.

   ii. The episode terminates when $s_{[t+n]}$ is the terminal state. This is described by the rule $sg\langle\text{ACT}\rangle \rightarrow ss'$, where $s' = s_{[t+n]}$ is the terminal state. s' is added to the end of the string and no new variable is on the right side of the rule because the episode terminates.

   iii. The iteration gets stuck in an infinite loop because the Actor never achieves g nor the terminal state is ever reached. This is described by the rule $sg\langle\text{ACT}\rangle \rightarrow sg\langle\text{ACT}\rangle$, indicating that the Actor continues to have control indefinitely.

The type of grammar that handles the above rules is an unrestricted grammar, which is formally defined by specifying (V, Σ, P, S), where V is a collection of variables (also called nonterminal symbols), Σ is the alphabet (also called terminal symbols), P are the rules (also called production rules), and S is the start variable (i.e., $S \in V$). Additionally, s and g are singular state or goal from the set $\mathcal{S}$ or $\mathcal{G}$ respectively.

Definition 1. Let β be a basic goal transformation function. A *basic grammar* for β is a 4-tuple (V, Σ, P, S) where $V = \{S, \langle\text{GOAL}\rangle, \langle\text{ACT}\rangle\}$ is the set of variables, $\Sigma = \mathcal{S} \cup \mathcal{G}$ is the set of constants, and P contains only the following production rules:

(a) For every possible starting state s, there is exactly one rule of the form: $S \rightarrow s\langle\text{GOAL}\rangle$.

(b) For every assignment $\beta(s, \_) \rightarrow g$, there is exactly one rule of the form: $s\langle\text{GOAL}\rangle \rightarrow sg\langle\text{ACT}\rangle$.

(c) For every tuple (s, g) where s is a nonterminal state, there is exactly one rule of one of the following forms:

   i. $sg\langle\text{ACT}\rangle \rightarrow ss'\langle\text{GOAL}\rangle$, where s' is a nonterminal state.

   ii. $sg\langle\text{ACT}\rangle \rightarrow ss'$, where s' is the terminal state.

iii. $sg\langle\text{ACT}\rangle \rightarrow sg\langle\text{ACT}\rangle$.

Example 1. This example shows a basic grammar for some basic goal transformation function. It generates the optimal trajectory in the 1-visit corridor (Figure 2 (a)), which explains how Basic GoalAct can solve this task. For clarity, we define that each goal g_i is achieved by reaching state s_i ($0 \leq i \leq 6$). s_0 is the terminal state.

Consider a basic grammar where $\Sigma = \{s_i, g_i \mid 0 \leq i \leq 6\}$, and P contains the following rules:

$$S \rightarrow s_3\langle\text{GOAL}\rangle \tag{1}$$
$$s_3\langle\text{GOAL}\rangle \rightarrow s_3g_6\langle\text{ACT}\rangle \tag{2}$$
$$s_6\langle\text{GOAL}\rangle \rightarrow s_6g_0\langle\text{ACT}\rangle \tag{3}$$
$$s_ig_j\langle\text{ACT}\rangle \rightarrow s_is_j\langle\text{GOAL}\rangle \quad (1 \leq i, j \leq 6 \wedge i \neq j) \tag{4}$$
$$s_ig_0\langle\text{ACT}\rangle \rightarrow s_is_0 \quad (1 \leq i \leq 6) \tag{5}$$
$$s_ig_i\langle\text{ACT}\rangle \rightarrow s_ig_i\langle\text{ACT}\rangle \quad (1 \leq i \leq 6) \tag{6}$$

Applying rules (1) to (5) as follows derives the string $s_3s_6s_0$ (the number on top of the arrow indicates which rule is applied). Rule (6) is not used in the derivation.

$$\begin{aligned} S &\xrightarrow{1} s_3\langle\text{GOAL}\rangle \\ &\xrightarrow{2} s_3g_6\langle\text{ACT}\rangle \\ &\xrightarrow{4} s_3s_6\langle\text{GOAL}\rangle \\ &\xrightarrow{3} s_3s_6g_0\langle\text{ACT}\rangle \\ &\xrightarrow{5} s_3s_6s_0 \end{aligned}$$

Proposition 1. If $G = (V, \Sigma, P, S)$ is a basic grammar and $\{s_1, s_2, s_3\} \subseteq \Sigma$, then G cannot generate a string that contains both s_1s_2 and s_1s_3.

The proof of Proposition 1 is in the appendix. Proposition 1 says that it is impossible for a basic grammar to generate a string where the same state is followed by two different states. For example, basic grammars cannot generate a string that contains $s_3s_6s_5s_6s_0$ as a substring. This substring represents the trajectory needed to solve the 2-visit corridor task (see Figure 2 (b)). Therefore, Basic GoalAct cannot generate this trajectory, and thus cannot solve this problem.

5.2 Expressive Power of Mnemonic GoalAct

Differing from Basic GoalAct, the Goal Selector in Mnemonic GoalAct receives as input a sequence of states instead a single state. The Goal Selector has a finite memory: the Goal Selector can recall a sequence of states of up to a certain length k. We define $\mathcal{S}^{\leq k} = \bigcup_{0 \leq i \leq k} S^i$, where S is the set of nonterminal states and k is a nonnegative integer. Given S, the i-th power of S, denoted

S^i, is the set of all strings obtained by concatenating i times strings in S.[4] Therefore, $\tilde{s} \in \mathcal{S}^{\leq k}$ represents a concatenation of up-to k states. In particular, we are interested in chronologically ordered sequences of states observed by the Goal Selector before the current time step. At a time step t, the Goal Selector performs a goal transformation function $\beta(s, \tilde{s}) \rightarrow g$. The Actor receives as input g and s and executes a series of actions from time t to a time step $t + n$ when either $g_{[t]}$ is achieved or the terminal state τ is reached. If the terminal state is not reached, the Goal Selector performs a goal transformation function $\beta(s_{[t+n]}, \tilde{s}) \rightarrow g_{[t+n]}$.

Similar to how we define basic grammars, we construct a new type of grammar that captures the expressive power of Mnemonic GoalAct and provide its formal definition afterward. The following are the kinds of rules needed to model Mnemonic GoalAct:

(a) There can be one or more starting states. For every starting state s, we add one rule of the form: $S \rightarrow s\langle\text{GOAL}\rangle$. S is the start symbol. $\langle\text{GOAL}\rangle$ is a variable representing the Goal Selector. This ensures that the first symbol appearing in the resulting state trajectory string is an starting state and control is then given to the Goal Selector.

(b) For every goal transformation $\beta(s, \tilde{s}) \rightarrow g$, we add one rule of the form: $\tilde{s}s\langle\text{GOAL}\rangle \rightarrow \tilde{s}sg\langle\text{ACT}\rangle$. $\langle\text{ACT}\rangle$ is a variable representing the Actor. These rules can be read as follows: if the agent is in state s, $\tilde{s}$ are the previous states visited, and the Goal Selector is in control, then goal g is generated and control is given to the Actor. Therefore g is added so it is next to s in the resulting string.

(c) The Actor receives a state-goal (s, g) and behaves the same way as in Basic GoalAct, so it has the same three classes of rules.

Definition 2. Let β be a mnemonic goal transformation function. A *k-mnemonic grammar* for β is a 4-tuple (V, Σ, P, S), where $V = \{S, \langle\text{GOAL}\rangle, \langle\text{ACT}\rangle\}$ is the set of variables, $\Sigma = \mathcal{S} \cup \mathcal{G}$ is the set of constants, and P contains only the following production rules:

(a) For every possible starting state s, there is exactly one rule of the form: $S \rightarrow s\langle\text{GOAL}\rangle$.

(b) For every assignment $\beta(s, \tilde{s}) \rightarrow g$ where $\tilde{s} \in \mathcal{S}^{\leq k}$, there is exactly one rule of the form: $\tilde{s}s\langle\text{GOAL}\rangle \rightarrow \tilde{s}sg\langle\text{ACT}\rangle$.

(c) For every tuple (s, g) where s is a nonterminal state, there is exactly one rule of one of the following forms:

i. $sg\langle\text{ACT}\rangle \rightarrow ss'\langle\text{GOAL}\rangle$, where s' is a nonterminal state.

ii. $sg\langle\text{ACT}\rangle \rightarrow ss'$, where s' is the terminal state.

iii. $sg\langle\text{ACT}\rangle \rightarrow sg\langle\text{ACT}\rangle$.

Example 2. This example shows a k-mnemonic grammar for some mnemonic goal transformation function. It generates the optimal trajectory in the 2-visit corridor (Figure 2 (b)), which explains how Mnemonic GoalAct can solve the task.

4. Formally: $S^0 = \{\lambda\}$, where λ is the empty string, and $S^{i+1} = \{wv \mid w \in S^i \text{ and } v \in S\}$. ($\lambda$ is the empty string.)

Consider a 3-mnemonic grammar where $\Sigma = \{s_i, g_i \mid 0 \leq i \leq 6\}$, and P contains the following rules:

$$S \to s_3\langle\text{GOAL}\rangle \tag{1}$$
$$s_3\langle\text{GOAL}\rangle \to s_3 g_6\langle\text{ACT}\rangle \tag{2}$$
$$s_3 s_6\langle\text{GOAL}\rangle \to s_3 s_6 g_5\langle\text{ACT}\rangle \tag{3}$$
$$s_3 s_6 s_5\langle\text{GOAL}\rangle \to s_3 s_6 s_5 g_6\langle\text{ACT}\rangle \tag{4}$$
$$s_3 s_6 s_5 s_6\langle\text{GOAL}\rangle \to s_3 s_6 s_5 s_6 g_0\langle\text{ACT}\rangle \tag{5}$$
$$s_i g_j\langle\text{ACT}\rangle \to s_i s_j\langle\text{GOAL}\rangle \quad (1 \leq i, j \leq 6 \wedge i \neq j) \tag{6}$$
$$s_i g_0\langle\text{ACT}\rangle \to s_i s_0 \quad (1 \leq i \leq 6) \tag{7}$$
$$s_i g_i\langle\text{ACT}\rangle \to s_i g_i\langle\text{ACT}\rangle \quad (1 \leq i \leq 6) \tag{8}$$

Applying rules (1) to (7) as follows derives the string $s_3 s_6 s_5 s_6 s_0$ (the number on top of the arrow indicates which rule is applied). Rule (8) is not used in the derivation.

$$\begin{aligned}
S &\xrightarrow{1} s_3\langle\text{GOAL}\rangle \\
&\xrightarrow{2} s_3 g_6\langle\text{ACT}\rangle \\
&\xrightarrow{6} s_3 s_6\langle\text{GOAL}\rangle \\
&\xrightarrow{3} s_3 s_6 g_5\langle\text{ACT}\rangle \\
&\xrightarrow{6} s_3 s_6 s_5\langle\text{GOAL}\rangle \\
&\xrightarrow{4} s_3 s_6 s_5 g_6\langle\text{ACT}\rangle \\
&\xrightarrow{6} s_3 s_6 s_5 s_6\langle\text{GOAL}\rangle \\
&\xrightarrow{5} s_3 s_6 s_5 s_6 g_0\langle\text{ACT}\rangle \\
&\xrightarrow{7} s_3 s_6 s_5 s_6 s_0
\end{aligned}$$

Proposition 2. Any basic grammar is a 0-mnemonic grammar.

Proposition 2 is trivially true because a basic grammar is defined to be equivalent to a 0-mnemonic grammar where $\tilde{s}$ is the empty string.

Basic grammars and k-mnemonic grammars correspond to the expressive power of basic and Mnemonic GoalAct respectively. Proposition 2 says that any basic grammar is itself a 0-mnemonic grammar. Proposition 1 implies that basic grammars cannot generate the string $s_3 s_6 s_5 s_6 s_0$, which contains both $s_6 s_5$ and $s_6 s_0$. Example 2 shows that there exists a k-mnemonic grammar that can generate this string. Therefore Mnemonic GoalAct is strictly more expressive than Basic GoalAct.

6. Experiments

To demonstrate that Mnemonic GoalAct is strictly more expressive than Basic GoalAct, we implement and test both systems in the following environments. We also reimplement h-DQN as a baseline.

- Corridor: the 2-visit corridor task (Figure 2 (b)). If the agent visits s_6 (asterisk) at least twice and then reaches s_0, it receives a reward of $+1$; otherwise, the reward is $+0.01$. Moving from s_5 to s_6 constitutes a visit to s_6. (Taking a right move in s_6 has no effect.) An episode ends after 20 time steps.

- Doom: the 2-visit corridor task recreated in a Doom map using the ViZDoom API (Kempka et al., 2016). The agent always faces right and observes $320 \times 240 \times 3$ RGB game frames. (i.e., an observed state is a screenshot of the game.)

- Grid: a navigation task in a 5×5 gridworld (Figure 4 (d)). The starting state is marked 0. The terminal state τ is above the first row. To obtain a reward of $+1$, the agent must visit landmark 1, return to 0, repeating the same procedure for landmarks 2 and 3, and finally reach τ. The visits can be intertwined, for example, $0\ 1\ 0\ 1\ 2\ 1\ 0\ 3\ 0\ \tau$ produces a reward of $+1$. In all other cases the reward is 0. An episode ends after 60 time steps. As per Theorem 1, Basic GoalAct cannot generate a trajectory that contains both $0\ 1$ and $0\ \tau$.

6.1 Implementation

As per our expressiveness analysis, basic and Mnemonic GoalAct differ only in their Goal Selectors. Both systems and h-DQN use a hardcoded optimal Actor $\gamma(s_a, g)$, which takes the actions that result in the shortest path from the current state s_a to a state that satisfies a given goal g. Therefore, any disparity in performance is due to their Goal Selectors.

The Mnemonic Goal Selector uses REINFORCE (Williams, 1992) to learn a parameterized stochastic policy $\pi(g|s, \tilde{s}; \theta)$, where g is the goal selected, s is the current observation, $\tilde{s}$ is a sequence of past observations, and θ is the set of parameters. The performance measure is the expected return $\mathbb{E}[G]$. θ is optimized by gradient ascent on $\mathbb{E}[G]$ in the direction $G\nabla \ln \pi(g|s, \tilde{s}; \theta)$. As shown in Figure 3, it uses a recurrent neural network (RNN) (Rumelhart et al., 1986) to learn temporal representations from variable-length sequences of states observed by the agent. When the observation dimension is high (e.g., images), it consists of two convolutional layers, followed by a layer of gated recurrent units (GRUs) (Cho et al., 2014), which is a specialized RNN that uses memory, and finally a dense softmax output layer, in which the number of units is equal to $|\mathcal{G}|$. When the observation dimension is low, the input is directly passed to a GRU layer followed by an output layer. The Basic Goal Selector replaces the GRU layer with a dense layer but is otherwise identical. The code of our implementation is available at `https://github.com/ospur/goal-reasoning`.

6.2 Training

Mnemonic GoalAct and Basic GoalAct use REINFORCE; random goals are selected in the initial 1000 episodes (exploration phase). h-DQN uses a one-step Q-learning algorithm with ϵ-greedy goal selection and experience replay (Schaul et al., 2015): ϵ decays from 1 to 0.01 over 15000 steps; the replay size is 100000; the batch size is 64; the target network update rate is 0.001. Other hyperparameters are shown in Table 1.

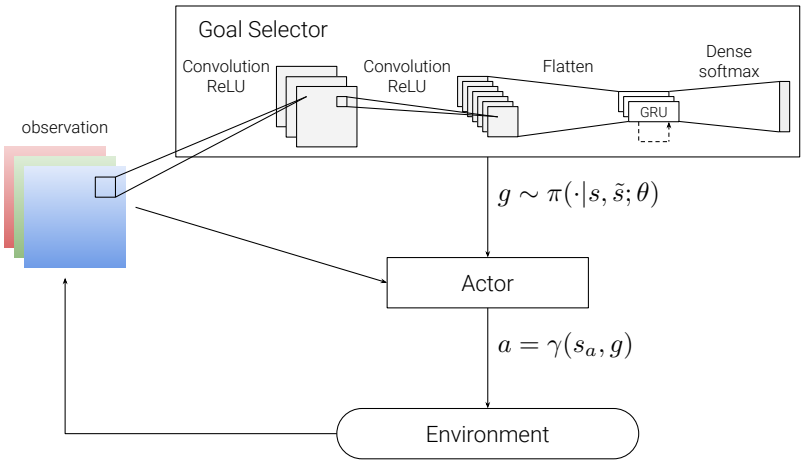


Figure 3: The architecture of Mnemonic GoalAct. The convolutional layers are not used when the observation dimension is low.

Table 1: Goal Selector hyperparameters.

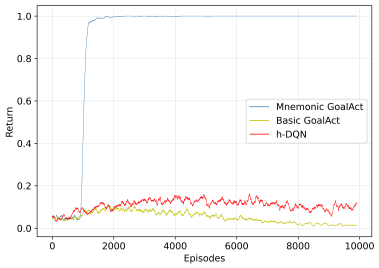
	Mnemonic GoalAct	Basic GoalAct	h-DQN
Corridor Grid	GRU: 64 units	Dense 1: 16 units [ReLU] Dense 2: 32 units [ReLU]	
Doom	Conv 1: 32 filters, (8, 8) kernel, strides=4 [ReLU] Conv 2: 64 filters, (4, 4) kernel, strides=2 [ReLU] GRU: 256 units	Dense: 256 units [ReLU]	
Output activation	softmax		linear
Loss	categorical crossentropy		Huber
Optimizer	Adam		RMSprop
Learning rate	0.001		

6.3 Results

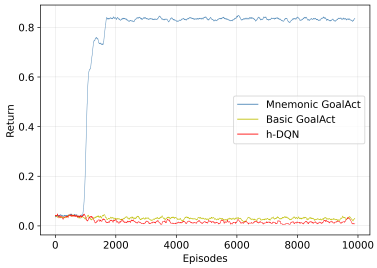
Mnemonic GoalAct outperforms Basic GoalAct and h-DQN in all the environments (Figure 4). In Corridor, Mnemonic GoalAct learns an optimal policy in 2000 episodes whereas Basic GoalAct and h-DQN achieve a return of 0.013 and 0.116 respectively after 10000 episodes. In Doom, Mnemonic GoalAct, Basic GoalAct, and h-DQN achieve a return of 0.835, 0.029, and 0.006, respectively. In Grid, Mnemonic GoalAct learns an optimal policy in 14000 episodes; Basic GoalAct and h-DQN are unable to find an optimal policy after 20000 episodes.

7. Related Work

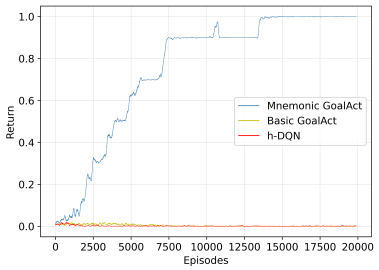
Whereas the GoalAct architecture is presented in the context of goal reasoning, the implementation of the architecture is related to the following lines of research.



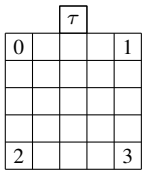
(a) Corridor



(b) Doom



(c) Grid



(d) Grid environment

Figure 4: (a) (b) (c) Learning curves of Mnemonic GoalAct, Basic GoalAct, and h-DQN. The results are the average of 10 runs smoothed by a 100-episode moving average. (d) The grid environment.

Deep RL. The use of neural networks for RL is inspired by DQN (Mnih et al., 2015), which combines a semi-gradient Q-learning algorithm with a deep convolutional neural network to learn to play Atari 2600 games. The Q-network provides the behavior policy and updates online. The target network, which is used to compute the temporal difference error, updates periodically and remains fixed between updates. An extension of DQN that uses a recurrent neural network is the deep recurrent Q-network (DRQN) (Hausknecht & Stone, 2015), which replaces the fully-connected layer in DQN with a layer of long short-term memory (LSTM) (Hochreiter & Schmidhuber, 1997). LSTM integrates state information through time. DRQN learns to play several partially observable versions of Atari games and achieves results comparable to DQN. Instead of Q-learning (Watkins & Dayan, 1992), our implementation uses a policy gradient method (Williams, 1992).

Options. Conceptually, the process of the Goal Selector giving control to the Actor to achieve a goal g is similar to the options framework (Sutton et al., 1999). An option is a temporally extended course of actions consisting of three components: an initiation set I, a policy π, and a termination condition β. An agent, controlled by a policy π', and it reaches a state s and $s \in I$. In that case o is selected, the agent takes actions according to π until o terminates according to β. At this point, control is given back to π'. Giving control to o when it reaches state s is comparable to the Goal Selector giving control to the Actor to achieve g and when the o terminates according to β is comparable to the Actor giving back control to the Goal Selector.

Hierarchical RL. The implementation of our architecture as a deep learning system is based on h-DQN (Kulkarni et al., 2016). A similar system is FuNs (Vezhnevets et al., 2017), which also has a two-level architecture but with two main differences: both levels use LSTM, another kind of RNN; the goals in FuNs are directional whereas in h-DQN the goals are absolute locations. The notion of goal used in our expressiveness analysis can handle both cases provided that a goal can be verifiably achieved in one or more states.

Memory-based RL. Utile Suffix Memor (USM) (McCallum, 1995), uses a suffix tree as a structural memory to deal with the problem of perceptual aliasing in RL, the situation where different states appear as the same observation to the agent. Observations are used to expand the USM suffix tree and compute the utility of the states as a function of the rewards obtained. It allows to consider rewards that are dependent on previous states visited. The difference versus our work is that in our implementation we use a recurrent neural network to provide memory capability rather than explicitly using a separate memory component.

Subgoal learning in planning. ActorSim (Bonanno et al., 2016) is a system that integrates goal reasoning and hierarchical planning. The agent is trained by indicating which goals to selected from particular observations (i.e., Minecraft's screenshots). It uses a CNN to learn which goal to select given a particular observation. In our work, we are using RL to learn from the state and the rewards. In one of our experiments, states are game's screenshots. The other difference is that in their system, training data is provided by an expert whereas in our case the agent is learning by interacting with the environment.

Expressive power of RNNs. Shallow networks have the same expressive power as RNNs (Khrulkov et al., 2018). In other words, any function approximated by an RNN can also be approximated by a shallow network. Nonetheless, a shallow network may require an exponential width increase to approximate the same function as an RNN, which can be intractable in many domains. In our work, we are not comparing the expressive power of RNNs and shallow networks. Instead, in the context of a two-level GoalAction architecture, we show strict expressive power difference between a framework that can remember states (using an RNN) and one without memory capability. Furthermore, we define the kinds of state sequences that can be generated by characterizing the grammars used to generate the sequences.

8. Concluding Remarks

In this paper we position the GoalAct architecture in the broader context of goal reasoning capabilities: first mapping its capabilities relative to the OODA framework and then on Aha (2018)'s goal dimensions. We then model its goal reasoning capabilities using formal grammars. We analyze the expressiveness of state trajectories of two existing variants: Basic and Mnemonic GoalAct. The use of a special type of unrestricted grammar allows us to directly model the architecture behavior using this type of grammar. We then prove that Mnemonic GoalAct is strictly more expressive than Basic GoalAct by characterizing the grammars generating the states sequences.

The experimental results are consistent with our expressiveness analysis. In the deterministic environments, the two Basic GoalAct systems are unable to generate the trajectories that produce

the maximum reward; in contrast, the Mnemonic GoalAct system is capable of learning to generate the trajectories and thus obtaining a much higher reward. Since the policies (i.e., softmax and ϵ-greedy) during training are not deterministic, it is possible for the Basic GoalAct systems to obtain a high reward by chance, but they are unable to do so consistently.

The need for the agent to remember which states it visited so far makes these environments non-Markovian. For example, in the corridor domain 2(b), the agent needs to remember that it has visited the asterisk location once so it can visit it twice to maximize its rewards. In principle, it would be possible to make the states Markovian by adding information into the states of all previously-visited locations. In fact, this is the purpose of adding an RNN layer. Our works shows what is the effect of the memory on the kinds of state sequences that can be generated.

For future work, we would like to focus on three lines of research. First, in order to further understand the architecture described in this work, we want to investigate the expressiveness of architectures that have additional recurrent levels on top of the Goal Selector. We conjecture that a single recurrent level with a sufficient number of units can simulate any number of recurrent levels and thus adding more levels will not increase its expressive power. Secondly, we want to integrate deep learning methods and state-of-art planning systems to enable a greater degree of autonomy that reduces the requirement of planning domain knowledge. The third is to consider the case when the goal selection process is stochastic (i.e., multiple goals can be selected based on the same input).

Acknowledgements

This research is supported by the Office of Naval Research grants N00014-18-1-2009 and N68335-18-C-4027 and the National Science Foundation grant 1909879 and Independent Research and Development (IR/D) Plan. 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 National Science Foundation.

References

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

Ambite, J. L., & Knoblock, C. A. (2001). Planning by rewriting. *Journal of Artificial Intelligence Research*, *15*, 207–261.

Bonanno, D., Roberts, M., Smith, L., & Aha, D. W. (2016). Selecting subgoals using deep learning in minecraft: A preliminary report. *IJCAI Workshop on Deep Learning for Artificial Intelligence*.

Boyd, J. (1995). Ooda loop. *Center for Defense Information, Tech. Rep*.

Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*.

Choi, D., & Langley, P. (2018). Evolution of the icarus cognitive architecture. *Cognitive Systems Research*, *48*, 25–38.

Cox, M., Alavi, Z., Dannenhauer, D., Eyorokon, V., Munoz-Avila, H., & Perlis, D. (2016). Midca: A metacognitive, integrated dual-cycle architecture for self-regulated autonomy. *Proceedings of the AAAI Conference on Artificial Intelligence*.

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

Erol, K., Hendler, J., & Nau, D. S. (1994). Htn planning: Complexity and expressivity. *AAAI* (pp. 1123–1128).

Hausknecht, M., & Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. *AAAI Fall Symposium* (pp. 29–37).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*, 1735–1780.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., & Jaśkowski, W. (2016). ViZDoom: A Doom-based AI research platform for visual reinforcement learning. *IEEE Conference on Computational Intelligence and Games* (pp. 341–348). IEEE.

Khrulkov, V., Novikov, A., & Oseledets, I. (2018). Expressive power of recurrent neural networks. *International Conference on Learning Representations*. https://openreview.net/pdf?id=S1WRibb0Z.

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

Kulkarni, T. D., Narasimhan, K., Saeedi, A., & Tenenbaum, J. (2016). Hierarchical deep reinforcement learning: integrating temporal abstraction and intrinsic motivation. *Advances in Neural Information Processing Systems* (pp. 3675–3683).

Laird, J. E. (2019). *The soar cognitive architecture*. MIT press.

McCallum, R. A. (1995). Instance-based utile distinctions for reinforcement learning with hidden state. In *Machine learning proceedings 1995*, 387–395. Elsevier.

Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529–533.

Mousavi, H. K., Liu, G., Yuan, W., Takáč, M., Muñoz-Avila, H., & Motee, N. (2019). A layered architecture for active perception: Image classification using deep reinforcement learning. *arXiv preprint arXiv:1909.09705*.

Rumelhart, D. E., Smolensky, P., McClelland, J. L., & Hinton, G. E. (1986). *Schemata and sequential thought processes in pdp models*, (p. 7–57). Cambridge, MA, USA: MIT Press.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Sipser, M. (2012). Introduction to the theory of computation. *Cengage Learning*.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, *112*, 181–211.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., & Kavukcuoglu, K. (2017). Feudal networks for hierarchical reinforcement learning. *International Conference on Machine Learning* (pp. 3540–3549).

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*, 279–292.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*, 229–256.

Appendix

Proposition 1. If $G = (V, \Sigma, P, S)$ is a basic grammar and $\{s_1, s_2, s_3\} \subseteq \Sigma$, then G cannot generate a string that contains both $s_1 s_2$ and $s_1 s_3$.

Proof. Assume that G can generate a string that contains both $s_1 s_2$ and $s_1 s_3$ and obtain a contradiction. The rules that add an s' to the right of an s are of the form (Definition 1 (c) i and ii):

$$sg\langle \text{ACT} \rangle \rightarrow ss'\langle \text{GOAL} \rangle \mid ss'$$

Since for every tuple (s, g) there is exactly one rule of the above form, and G can generate both $s_1 s_2$ and $s_1 s_3$, P must contain

$$s_1 g_a \langle \text{ACT} \rangle \rightarrow s_1 s_2 \langle \text{GOAL} \rangle \mid s_1 s_2 \tag{1}$$
$$s_1 g_b \langle \text{ACT} \rangle \rightarrow s_1 s_3 \langle \text{GOAL} \rangle \mid s_1 s_3 \tag{2}$$

where $g_a \neq g_b$.

Since $s_1 s_2$ and $s_1 s_3$ are generated by substituting the left-hand side of rules (1) and (2), G must be able to generate an intermediate string that contains both $s_1 g_a$ and $s_1 g_b$. The rules that add a g to the right of an s are of the form (Definition 1 (b)):

$$s\langle \text{GOAL} \rangle \rightarrow sg\langle \text{ACT} \rangle$$

Hence P must contain

$$s_1\langle \text{GOAL} \rangle \rightarrow s_1 g_a \langle \text{ACT} \rangle \tag{3}$$
$$s_1\langle \text{GOAL} \rangle \rightarrow s_1 g_b \langle \text{ACT} \rangle \tag{4}$$

Since there is exactly one rule of the form $s_1\langle \text{GOAL} \rangle \rightarrow s_1 g\langle \text{ACT} \rangle$, the equality $g_a = g_b$ must be true, which contradicts $g_a \neq g_b$. Therefore the original assumption is false.

□