Monitoring GDA Agent's Expectations Generated from Durative Effects

Author Names Withheld

Abstract. One of the crucial capabilities for robust agency is self-assessment, namely the capability of the agent to compute its own boundaries. A method of assessing these boundaries is using so-called expectations: constructs defining the boundaries of an agent's courses of action as a function of the plan, the goals achieved by that plan, the initial state, the action model and the last action executed. In this paper we redefine four forms of expectations from the goal reasoning literature but, unlike those works, the agent reasons with durative actions. We present properties and a comparative study highlighting the trade offs between the expectations.

1 Introduction

There has been an increasing interest in AI safety, creating reliable AI agents that perform within their boundaries. With the increasing sophistication of autonomous systems, situations arise in which unexpected situations may occur. This happens when the agent and/or the environment in which the agent is operating behaves in a way that is inconsistent with the agent's planning knowledge.

Goal-driven autonomy (GDA) agents supervise the agent's execution of its current plan and formulate new goals when discrepancies arise between the agent's **expectations** of the actions in the plan executed so far and the resulting state. To detect discrepancies, the agent generates a set of expectations $X(\pi, a)$, as a function of the next action a in the plan π to be executed. The agent can then check if this expectation is satisfied in the state, s.

Naively, for computing the expectation $X(\pi, a)$ it would seem sufficient for the agent to simply check if the preconditions of a are satisfied in s (i.e., to define $X(\pi, a)$= *"the preconditions of a"*). However an agent using expectations defined this way would not be checking the plan trajectory in any way. Similarly, we could project the state s before the action a is the be executed from the initial state s_0 using π. (i.e., to define $X(\pi, a) = s$). Expectations are often generated this way in the goal reasoning literature [1],[2],[3]). The issue here is that there are often cases where there are variables in the state that have no bearing on π and thus if they are altered it will not impact the agents execution. To define the expectations over the entire state would cause discrepancies in situations where they are not needed. Researchers have observed that the notion of expectations plays a key role in the resulting performance of goal reasoning agents [4, 5].

In this paper we report on our studies computing expectations when the actions are durative. This means actions have continuous effects over some segment of time. For instance, the agent may control a camera to follow a target while turning around 20° on a swivel. Performing this action requires some time to complete as a function of the rotation speed of the camera. Furthermore, while performing this action, the agent may initiate another action such as zooming out by a 1/3 zoom ratio. This action itself requires time to complete and may be initiated while the previous action is still not completed. Therefore, expectations are also a function of the time t and not of a specific action in π since multiple actions may have been initiated. Thus, in our work, expectations are a pairs of the form $X(\pi, t)$, where t is some time after π was initiated.

The following are the main contributions of this paper, centered around the notion of expectations when GDA agents perform durative actions:

- We re-define the notion informed, regression [6] and goldilocks [7] expectations.
- We formulate properties on regression expectations providing guarantees on the success of the remaining plan when certain conditions are met.
- We provide a empirical evaluation on two very different domains and discuss trade offs between the different forms of expectations.

2 Related Work

GDA is a goal reasoning model in which agents monitor the current plan's execution and assess whether the observed states match the agents' own expectations. GDA agents formulate a plan π achieving goals g from the current state s. They also formulate the expectations $X(\pi, a)$ for every action a in π. As each action a in π is executed, the agent checks if its expectations $X(\pi, a)$ are satisfied in the current state s. When they are not satisfied the GDA agents follows a procedure to formulate new goals g', a new plan π', new expectations $X(\pi', a)$ and the process repeats itself.

Research on GDA agents have explored different facets of the cycle including generating explanations for the mismatch between the agent's expectation and the state [8] and procedures to generate new goals [9]. In this discussion, we focus on work formulating the expectations. A variety of formulations for the expectations have been formulated mostly for symbolic domains. This includes defining the expectations as [10]:

- (1) immediate expectations: checking the preconditions of the next action to execute;
- (2) state expectations: the projected state by applying the actions in the plan executed so far;
- (3) informed expectations: the cumulative effects of the actions executed so far;
- (4) goal regression expectations: starting from the goals, accumulating the conditions in the state necessary to execute the rest of the plan, building on classical work [11];
- (5) goldilocks expectations [7] combining (3) and (4).

In our work we re-define informed, regression, and goldilocks expectation's when actions have durative effects.

GDA expectations have been explored for actions with numeric fluents. Intervals $(x_\downarrow, x_\uparrow)$ are used to indicate valid values for a variable. Actions define a function $f(x)$ indicating new values for x after the action is applied. Like their symbolic counterparts, these works assume the actions in a plan π are executed instantaneously and in sequence. Therefore, expectations are also denoted as a function $X(\pi, a)$ of each action $a \in \pi$. In Wilson et. al. [12], state expectations are defined in which the intervals are projected forward $(f(x_\downarrow), f(x_\uparrow))$ for each action. [13] extends these ideas for informed, regression, and goldilocks expectations. In our work, we re-define when the numeric effects of the action change over time (e.g., $f(x, t)$) and include situations when actions in the plan π are performed concurrently. Hence, we define expectations over time, $X(\pi, t)$.

Studies on expectation failures emanate from the plan execution literature [14]. For instance, [15] proposed a model to find the reason for a failure in the plan. [16] propose a taxonomies of expectation failures as a function of the plan, planning knowledge, the resulting state and the state expectations. For example, the incorrect domain knowledge class refers to expectations failures caused by planning operators incorrectly modeling the actual operators behavior. [17] presents an alternative taxonomy of failures related to the execution of the SIPE HTN planner. For instance, a failure maybe attributed to a condition that was held to be true at planning time but it is no longer true when the plan is executed. [18] proposes execution failures when the plan doesn't meet quality considerations. None of these works consider failures due to durative actions.

3 Preliminaries

Throughout this paper, we will use partial mappings. A partial $f : A \nrightarrow B$ indicates a function that is defined only for a subset of A. When referring to the set of variables defined in a mapping, we write A_f, meaning the set of variables from A defined in the mapping f. When listing the entire mapping (e.g., in examples), we will use a dictionary format to represent the partial mapping, where the keys are the variables, and the values are what they map to. For example, $f = \{a : 1, b : 0\}$ denotes that there are two variables in mapping f: a and b, and that their respective values, denoted $f[a]$ and $f[b]$, are 1 and 2.

Since we are dealing with actions that have functional effects, such as changes over a time t, we use lambda calculus to represent them. Briefly, functions are represented as tuples. The first element in the tuple is the function to be exercised, all following elements in the tuple are arguments to that function. For example, $(-\ 3\ 2)$ represents the minus function, where 3 is the first argument and 2 is the second, i.e., 3 - 2. Therefore $(-\ 3\ 2)$ would return 1. There can also be functions with free variables, called lambda functions. We use these to represent functions dependent on a time variable. For example, one can write $f = \lambda t.(-\ t\ 3)$ to represent a function with a singular argument t. This function f thus returns the argument given it subtracted by 3, thus $(f\ 5)$ would return 2.

A state is a mapping $S : V \to \mathbb{R}$ from a collection of variables V to a collection of real numbers $\mathbb{R}$. Since we are dealing with real numbers, its unrealistic to always know the exact values of variables [19]. Therefore we represent the value of a

variable, e.g., at-y$[r]$, with two mappings denoting its lower and upper bounds, e.g., (at-y$[r]_{\downarrow}$, at-y$[r]_{\uparrow}$). Actions have durative effects, meaning for a time period t the variable is changing as a function of t. Table 2 shows an example of a state, while Table 1 shows an example operator that can be applied to this state. As exemplified in Table 1, an operator is a 4-tuple *o=(name parameters* pre^a eff^a*)*.

A set of goals $\mathcal{G}$ is a partial mapping $\mathcal{G}$: $V \nrightarrow \mathbb{R}$ from a collection of variables V to a collection of real numbers $\mathbb{R}$. These are also represented with two mappings to denote an upper and lower bound for each variable in V_G.

An operator's preconditions are a list of variables boundings. They are represented as the partial mapping $pre^a : V \nrightarrow C$ of variables to constants (with individual variables for upper and lower bounds). For example, in the operator $move\_north$ shown in Table 1 we are checking that the lower bound of the variable at-y$[x]$ (i.e., at-y$[x]_{\downarrow}$) is $(* \; t$ move-rate$[x]_{\uparrow}))$.

We define the set of effects from an action as a partial mapping $eff^a : V \nrightarrow \Lambda$ from variables to lambda functions. Looking at Table 1, we can see that the effects are a list of 3-tuples. For the purpose of our calculations, we care solely about the functional changes to the state variables. Thus for all variables $v \in e$ where e is the effect list, $eff^a[v] = e[v][2][0]$. For example, from Table 1, one of the effects is "[at-y$[x]_{\downarrow} \rightarrow$ $(+$ at-y$[x]_{\downarrow}$ $((f_1 \; x) \; t)))$,". Thus eff^a[at-y$[x]_{\downarrow}$] would be the lambda function returned by "$(f_1 \; x)$" (as defined in the operator).

Applying operators change the values of variable as a function of time t. For example, consider applying the operator **move_north** (Table 1) to the initial state defined in Table 2 with arguments $(r, 2)$, indicating rover r is to move north for 2 time steps. The operator changes the rover's fuel level *fuel[r]* and its y coordinate, *at-y[r]*. The upper bound of its fuel level, at-y$[x]_{\uparrow}$, changes from 10 to $(+$ 10 ($s{-}t$ .9); when $t = 2$, the execution of the operator is completed and the value of at-y$[x]_{\uparrow}$ is set to 8.2 (i.e., $(+$ 10 $(* -2$ .9)).

A plan π is defined as a set of time stamped actions of the form (time, action). For example, in Table 2 the first action in π is (0, move_north(r, 2)), indicating that at time 0, the action move north with the parameters r and 2 is executed. The next action is (0, move_east(r, 2)). Since its starting time is also 0, the effect of this is the rover moving diagonally.

We denote the partial mapping $T_s : T \nrightarrow \mathcal{P}(A)$ as a mapping from a time t to the set of actions $T_s(t)$ starting at time t. Analogously, we also define the partial mapping $T_e : T \nrightarrow \mathcal{P}(A)$ as a mapping from times to sets of actions that end at those times ($\mathcal{P}(A)$ is the power set of A).

A planning problem is a triple $P = (S_0, A, \mathcal{G})$. A plan π solves a problem P if the following conditions are met: there is a sequence of states $S_0, S_1, ..., S_n$ such that: (1) S_i yields S_{i+i} in π. (2) $\mathcal{G}$ is satisfied in S_n.

S_i yields S_{i+1} by adding $\sum^a f_a(1)$ for all actions a such that $a \in T_s(t') \cup T_e(t'')$ with $t' \leq i \leq t''$, with f_a being the functional effect that changes a variable v in action a. That is, for each variable v, $S_{i+1}[v] = S_i[v] + \sum^a f_a[v](1)$.

A mapping of variables to values such as $\mathcal{G}$ is satisfied in a state S if for all $v_{\downarrow} \in V_g$, $G[v_{\downarrow}] \geq S[v_{\downarrow}]$ and for all $v_{\uparrow} \in V_g$, $G[v_{\uparrow}] \geq S[v_{\uparrow}]$. The same definition applies to an

(:operator move_north
:parameters x t
:condition
at-y$[x]_{\downarrow} \rightarrow (* \; t$ move-rate$[x]_{\uparrow})$
fuel$[x]_{\downarrow} \rightarrow (* \; t$ fuel-rate$[x]_{\uparrow})$
:effect
[at-y$[x]_{\downarrow} \rightarrow (+$ at-y$[x]_{\downarrow} \; ((f_1 \; x) \; t))),$
at-y$[x]_{\uparrow} \rightarrow (+$ at-y$[x]_{\uparrow} \; ((f_2 \; x) \; t))),$
fuel$[x]_{\downarrow} \rightarrow (+$ fuel$[x]_{\downarrow} \; ((f_3 \; x) \; t))),$
fuel$[x]_{\uparrow} \rightarrow (+$ fuel$[x]_{\uparrow} \; ((f_4 \; x) \; t)))]$
$f_1 = \lambda x.(\lambda t.(* \; t$ move-rate$[x]_{\uparrow})),$
$f_2 = \lambda x.(\lambda t.(* \; t$ move-rate$[x]_{\downarrow})),$
$f_3 = \lambda x.(\lambda t.(* \; -t$ fuel-rate$[x]_{\uparrow})),$
$f_4 = \lambda x.(\lambda t.(* \; -t$ fuel-rate$[x]_{\downarrow}))$

Table 1. Example of operator with a numeric fluent (fuel).

expectation X, which is also a mapping from variables to values, to be satisfied in a state S.

Similarly, let π_t be the portion of the plan π that remains to be executed at time $t \leq n+1$. That is, π_t includes all actions in π not in $T_e(0) \cup T_e(1) \cup \ldots T_e(t)$ (i.e., actions that are still under execution or whose execution has not started yet). π_t is valid if there is a sequence of states $S_t, S_{t+1}, ..., S_n$ yielded such that: (1) S_i yields S_{i+i} in π. (2) $\mathcal{G}$ is satisfied in S_n. In our work, we assume the state persists unmodified after the completion of the plan π, meaning $S_n = S_{n+1}$ and therefore, the empty plan $\pi_{n+1} = ()$ solves $(S_{n+1}, A, \mathcal{G})$ whenever π solves $(S_0, A, \mathcal{G})$.

4 Two Basic Operations

We introduce two basic operations $\oplus_S$ and $\ominus_P$, which are used to define precisely the different forms of expectations. Informally, $\oplus_S$ compounds lambda functions (useful for adding together effects of actions), whereas $\ominus_P$ removes a function from a compounded set (useful for removing the effect of an action as its function of t after it is completed).

We define $D = A \oplus_S^{t'} B$, where A are some variables (e.g., accumulated changes while compounding functions), S is the current state, t' is the current time, and B are the effects of some action (e.g., the next action in the plan). More generally, for any partial functions A and B, any time t', and any state S with $A : V \nrightarrow \Lambda$, $S : V \rightarrow \Lambda$, $t' \in Z^+$, and $B : V \nrightarrow \Lambda$, $A \oplus_S^{t'} B$ is a partial mapping $D_{t'} : V \nrightarrow \Lambda$ defined as follows:

1. if $v \in V_A \cap V_B$ where $A[v] = M$ and $B[v] = N$, then $D_{t'}[v] = \lambda t.(+ \; (M \; t) \; (N(- \; t \; t')))$.
2. if $v \in V_A - V_B$ then $D_{t'}(v) = A(v)$.
3. if $v \in V_B - V_A$ where $S[v] = M$ and $B[v] = N$, then $D_{t'}[v] = \lambda t.(+ \; (M \; t) \; (N \; (- \; t \; t')))$.

(:Initial State
{fuel : {r$_\downarrow$: 10, r$_\uparrow$: 10]}}
{Beacon_fuel : {r$_\downarrow$: 1, r$_\uparrow$: 1]}}
{at-y : {r$_\downarrow$: 2, r$_\uparrow$: 2, Beacon1$_\downarrow$: 0, Beacon1$_\uparrow$: 0}}
{at-x : {r$_\downarrow$: 0, r$_\uparrow$: 0, Beacon1$_\downarrow$: 2, Beacon1$_\uparrow$: 2}}
{lit : {Beacon1$_\downarrow$: 0, Beacon1$_\uparrow$: 0}}
{fuel-rate : {r$_\downarrow$: .9, r$_\uparrow$: 1.1}}
{move-rate : {r$_\downarrow$: .9, r$_\uparrow$: 1.1}}
:Actions
move_north, move_south, move_east, move_west,
light_beacon
:Goals
{lit : {Beacon1$_\downarrow$: 1, Beacon1$_\uparrow$: 1]}}
:Plan π
(0, move_north(r, 2)), (0, move_east(r, 2)),
(2, light_beacon(r, 1))

Table 2. Planning problem and a solution plan

4. for all other variables $D_{t'}$ is undefined (i.e., $V_{D_{t'}} = V_A \cup V_B$)

Informally, $A \oplus_S^{t'} B$ results in a function addition $(+\ (A[v]\ t)\ (B[v]\ (-\ t\ t')))$ when the variable v is defined in A and B (Case 1), or $(+\ (S[v]\ t)\ (B[v]\ (-\ t\ t')))$ when v is defined in B but not A (Case 3). We are using an updated time variable $(-\ t\ t')$ for the functions from B since these represent the effects of the next actions to be added into the expectation set. Since we are in the middle of the plan, we need to shift the value of t to represent that that action isn't starting at time $t = 0$. If the variable v is defined in A but not B, it's assigned $A[v]$ (Case 2). When it's undefined in A and B, then it's left undefined (Case 4). For example, if A, S, and B are defined as:

- $A = \{a : \lambda t.(+(*\ 2\ t)3), c : \lambda t.(t)\}$
- $S = \{a : \lambda t.(+(*\ 2\ t)3), b : \lambda t.(-(*\ 3\ t)4), c : \lambda t.(t)\}$
- $B = \{a : \lambda t.(*\ 2\ t), b : \lambda t.(t)\}$
- $t' = 3$ (current time is 3)
- Then $D = A \oplus_S^3 B = \{a : \lambda t.(-(*\ 4\ t)3), b : \lambda t.(-(*\ 4\ t)7), c : \lambda t.(t)\}$

In the resulting partial function $D[a] = \lambda t.(-\ (*\ 4\ t)\ 3)$ is obtained by adding the functions $(A[a]\ t)$ and $(B[a]\ (-t\ 3))$ (i.e., Case 1). This procedure is shown in Figure 1. $D[b]$ and $D[c]$ are similarly obtained from the rules defined in the $\oplus$ operator.

We define $D = A \ominus_P^{t'} B$, where A are some variables (e.g., accumulated changes while compounding functions), P are the preconditions from some action (e.g., the current action we are regressing from) and B are the effects of the action. More generally, let $A : V \nrightarrow \Lambda$, $P : V \nrightarrow C$, $t' \in Z^+$, and $B : V \nrightarrow \Lambda$, we define $A \ominus_P^{t'} B$ as a partial mapping $D : V \nrightarrow \Lambda$ with:

1. if $v \in V_A - V_B$ then $D[v] = A[v]$.
2. if $v \in V_A \cap V_B$ where $A[v] = M$ and $B[v] = N$, then $D[v] = (-\ (M\ t)\ (N\ (-\ t\ t')))$.

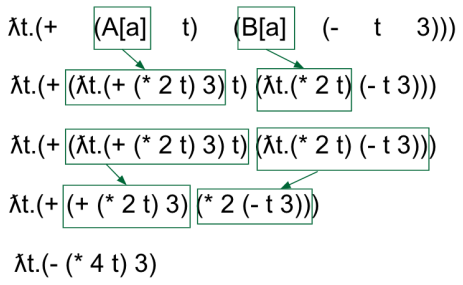

Fig. 1. Example calculation of D[a] from the $\oplus$ operator example

3. if $v \in V_P$ then $D[v] = \lambda t.(P[v])$
4. for all other variables D is undefined (i.e., $V_D = V_A \cup V_P$)

Informally, $A \ominus_P^{t'} B$ results in a new partial mapping that is defined for all variables from A and P. The new mapping takes the value $A(v)$ if v is defined in A but not in B (Case 1). If a variable v is defined in A and B, the new mapping takes the values after subtracting $(- (A[v]\ t)\ (B[v]\ (-\ t\ t')))$ (Case 2), If a variable v is defined in P the new mapping takes the value $P[v]$ (Case 3). If a variable is not defined in either A or P, it is left undefined (Case 4). For example, if we have the three partial functions A, P, and B, as follows:

- $A = \{a : \lambda t.(+\ t\ 3), b : \lambda t.(-\ (*\ 2\ t)\ 4)\}$
- $P = \{c : -2\}$
- $B = \{b : \lambda t.(*\ 2\ t)\}$
- $t' = 3$ (current time is 3)
- Then $D = A \ominus_P^3 B = \{a : \lambda t.(+\ t\ 3), b : \lambda t.(2), c : \lambda t.(\text{-}\ 2)\}$.

5 Informed Expectations with durative effects

Agents using Informed Expectations check that the compounded and accumulated effects are valid in the environment. Informally, informed Expectations accumulate a set of functions over time extracted from the effects of all previous durative actions executed so far in π. They compound all active durative actions' functions, and retain all past changes made to the state by actions that have finished executing.

Formally, informed expectations are denoted as $X_{inf}(\pi, \text{-}1, t, \emptyset) = inf_t$, for some time t. Each inf_t is recursively generated as follows:

1. $inf_{\text{-}1} = \emptyset$. (we start with $t =$ -1 for bookkeeping)
2. For all $t \geq 0$, inf_t is generated by the following 3 steps:
   (a) $inf_t = inf_{t-1}$
   (b) for all $a \in T_s(t)$, $inf_t = inf_t \oplus_{s_{t-1}}^t eff^a$.
   (c) for all $a \in T_e(t)$, $inf_t = inf_t \ominus_{\{\}}^t eff^a$

Case 1, the base case, indicates that before the first action is executed, we have no accumulated effects. The 3 steps of Case 2, the recursive case, are as follows: we start with the expectations computed up to time $t-1$ (Step 2 (a)). We add the effects of actions starting in t (Step 2 (b)). Finally, we substract the effects of actions terminating at time t (step 2(c)).

Example: If we are at time $t = 2$ in our plan, we can calculate inf_2 for $fuel$ as follows in Figure 2: The first step in caclulating the expectations is to combine inf_1 (Informed expectations at time 1) with eff^{move_north} (effects of the move north action) using the $\ominus^2_{\{\}}$ operator. Line 3 of the figure shows the substitutions of the values of these mappings (Line 2) applied using the $\ominus$ operator definitions. The right side of line 3 shows the simplified result of this operator application. Lines 4 and 5 sets up the second $\ominus$ operator, using the result of the last calculation and the effects of the move east action. Then, the left side of line 6 shows application of the operator, with the right side being the simplified result.

We can see the entire calculation in Figure 3. This figure includes all steps and for all variables and follows a similar calculation path as the simplified figure.

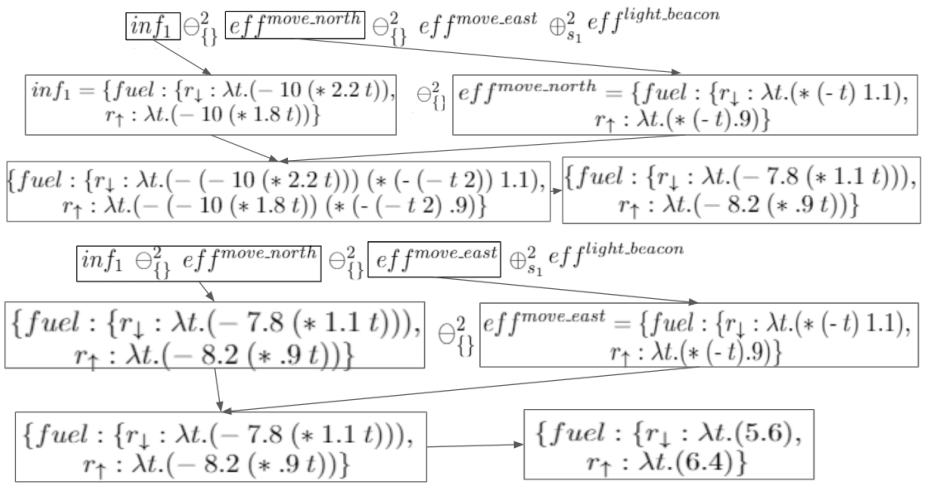

Fig. 2. Calculation of the informed expectations for variable $fuel$ at time step $t = 2$. The last operation step is left out because eff^{light_beacon} doesn't alter the variable $fuel$ and thus doesn't alter the expectation set for this variable. Full expansion of all variables in the state can be seen in Figure 3.

6 Regression Expectations

Informally, Regression Expectations accumulate a set of functions starting backwards from the last action in π. It compounds all active durative actions' inverse functions as

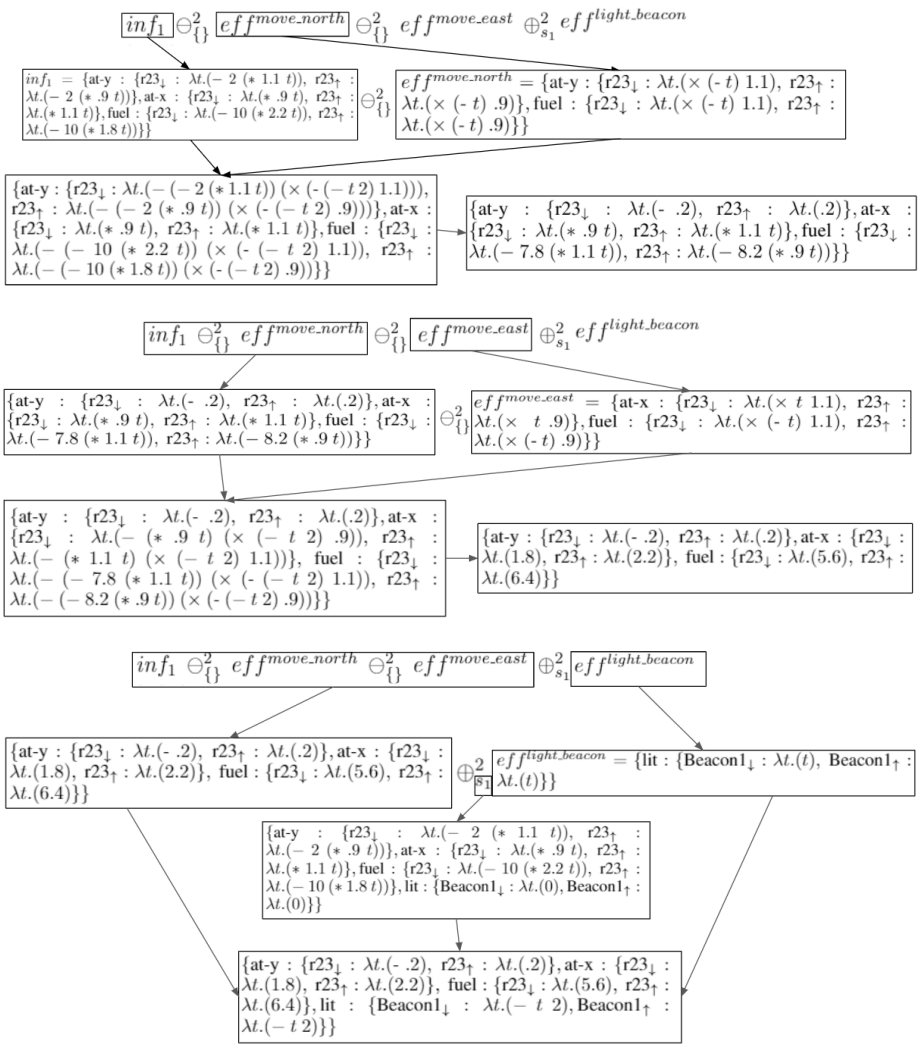

Fig. 3. Expanded calculation of the informed expectations at time step $t = 2$.

well as their preconditions, ensuring that the goals will still be met after the completion of π.

Formally, let n be the time step when π finalizes its execution, 0 the time when π starts its execution, t a natural number with $0 \leq t \leq n + 1$, we denote the regressed expectations at time t with $X_{reg}(\pi, t, n{+}1, \mathcal{G}) = reg_t$. Each reg_t is generated as follows:

1. $reg_{n+1} = \mathcal{G}$. (we start with $t = n + 1$ for bookkeeping)
2. For all $t \leq n$, we compute reg_t in three steps:
   (a) $reg_t = reg_{t+1}$

(b) for all $a \in T_s(s)$, $reg_t = reg_t \ominus^t_{pre^a} eff^a$
(c) for all $a \in T_e(t)$, $reg_t = reg_t \oplus^t_{s_{t+1}} eff^a$.

Case 1 is the base case; $t = n$ is the time that the last action in π completes its execution. So $t = n+1$ is the first time step after the completion of the plan's execution and we expect the set of goals $\mathcal{G}$ to be satisfied. If the goals are unknown, then $\mathcal{G} = \{\}$. The 3 steps of Case 2, the recursive case, are as follows: we start with the regressed expectations computed up to time $t + 1$ (Step 2 (a)). We subtract the effects of actions starting at time t (step 2(b)). Finally, we add the effects of actions ending in t (Step 2 (c)).

Agents using Regression Expectations check that the rest of the plan can be executed, and when finished the set of goals $\mathcal{G}$ (if they exist) will be satisfied.

Example: If we are at time $t = 2$ of the plan trace π in Table 2 we can calculate the Regression Expectations $reg_2 = X_{regress}(\pi, s_2, \mathcal{G})$ as follows (The preconditions and effects for $move\_east$ and $light\_beacon$ have not been shown before):

- $reg_4 = \mathcal{G} = \{\text{lit} : \{\text{Beacon1}_\downarrow : 1,\ \text{Beacon1}_\uparrow : 1]\}\}$. (i.e., Case 1 with $n = 3$)
- $reg_3 = reg_4 \oplus^3_{\{\}} eff^{light\_beacon} = \{\text{lit} : \{\text{Beacon1}_\downarrow : (+\ 1\ (-\ t\ 3)),\ \text{Beacon1}_\uparrow : (+\ 1\ (-\ t\ 3))]\}\}$ (i.e., Step 2 (b): light_beacon ends at $t = 3$).
- Thus $reg_2 = reg_3 \ominus^2_{pre^{light\_beacon}} eff^{light\_beacon} \oplus^2_{\{\}} eff^{move\_north} \oplus^2_{\{\}} eff^{move\_east}$ (i.e., Step 2 (b): light_beacon starts at $t = 2$ and Step 2 (c): move_east and move_north end at $t = 2$).

7 Goldilocks Expectations

Goldilocks Expectations [7] combines Informed and Regression Expectations. Formally, we define Goldilocks Expectations as $X_{gold}(\pi, t, \mathcal{G}) = gold_t$, where $gold_t = (inf_t, reg_t)$. That is, for ever time t, $gold_t$ is the pair containing the Informed and Regression Expectations for that time. An agent using $X_{gold}(\pi, i, \mathcal{G})$ checks the overlap of the regressed and the informed intervals, $[left(v'), right(v')] \cap [left(v"), right(v")]$. This ensures completing the goals while checking for inferred considerations from the action model such as efficiency.

Example: If we are at time $t = 2$, then we can compute the Goldilocks Expectations $gold_2 = (inf_2, reg_2)$

8 Property of Regression

Theorem 1. Let π be a plan solving (S$_0$, A, $\mathcal{G}$). If $X_{reg}(\pi, t, n + 1, \mathcal{G})$ is satisfied in S_t then π_t solves $(S_t, A, \mathcal{G})$.

Base case: t = n+1. At time $n + 1$, $X_{reg}(\pi, n + 1, n + 1, \mathcal{G}) = \mathcal{G}$ by definition. Therefore if $X_{reg}(\pi, n + 1, n + 1, \mathcal{G})$ is satisfied, then the empty plan $\pi_{n+1} = ()$ solves $(S_{n+1}, A, \mathcal{G})$.

Recursive case: We show that if reg_i is satisfied in $t = i$, then when executing one time step in π_i, reg_{i+1} will be satisfied in $t = i + 1$. By induction hypothesis, π_{i+1} solves $(S_{i+1}, A, \mathcal{G})$ and, hence, π_i solves $(S_i, A, \mathcal{G})$.

The calculation for reg_i begins with reg_i equaling reg_{i+1}. We analyze the three kinds of actions with respect to reg_i, and its such that if for each of these types of actions reg_i yields reg_{i+1} individually, then reg_i yields reg_{i+1} overall:

- **Actions that are continuing through time set** i: that is, each action a such that $a \in T_s(t') \cap T_e(t'')$ and $t' < i < t''$. For each such action a, reg_i will yield reg_{i+1} over the variables affected by a. That is, $reg_{i+1}[v] = reg_i[v] + \sum^a f_a[v](1)$.
- **Actions that end in time step** i. That is, each action a with $a \in T_e(t)$. For each such action a, its effects are added into the regression set as a function of time using the $\oplus$ operator. Specifically, this will execute step 1 of the $\oplus$ operator resulting in $reg_i = (+ \ (reg_{i+1} \ t) \ (eff^a \ (- \ t \ t`)))$. For this time step where the action is ending, $t = t'$, thus $(eff^a \ (- \ t \ t`)) = 0$ and $(+ \ (reg_{i+1} \ t) \ (eff^a \ (- \ t \ t`))) = reg_{i+1}$
- **For actions that begin in time step** i. That is, each action a with $a \in T_s(t)$. For each such action a, we are removing their effects from the regression set using the $\ominus$ operator. We consider first the effects and then the preconditions:
  - **Effects:** Specifically we use the second clause of the operator resulting in $reg_i = (- \ (reg_{i+1} \ t) \ (eff^a \ (- \ t \ t')))$. For this time step where the action is starting, $t = t'$, thus $(eff^a \ (- \ t \ t`)) = 0$ and $(- \ (reg_{i+1} \ t) \ (eff^a \ (- \ t \ t`))) = reg_{i+1}$
  - **Preconditions:** Lastly, we add in the preconditions for all actions that begin at time i. By adding these values into the regression set, we ensure that all actions beginning at time i will be executable.

9 Empirical Evaluation

We did a comparative study of the 5 different types of expectations across 2 temporal domains. The 5 Expectation types we tested were Immediate, Informed, Regression, Goal Regression, and Goldilocks Expectations (Immediate expectations in this case are simply checking the preconditions of the action before execution). The difference between goal regression and regression is that in the latter $\mathcal{G} = \emptyset$, accounting for situations when the goals are not known (e.g., using a domain-specific planner with implicit goals). For planning purposes, we use the Pyhop HTN planner [20], which was extended to handle numeric fluents over a temporal space and the HTN methods configured to generate correct plans. Other than the expectation type, the agent uses the same planning and discrepancy handling processes. Whenever a discrepancy is observed from the expectations, we use a simple goal-reasoning process to select the original goal and plan again from the current state. Thus, any performance changes is attributable to the expectations.

Marsworld Definition. We used is a temporal variant on the Marsworld domain [5], itself inspired by Mudsworld [21]. The agent has to navigate a 10x10 grid to turn on 3 randomly placed beacons. The grid space is continuous in this version. Each movement action drains some amount of the agents fuel. There is a second fuel resource which is used solely for lighting the beacon.

While executing its actions, the agent may unexpectedly have damage caused to it, forcing it to use more fuel per action until repaired (this can occur with a 2% probability

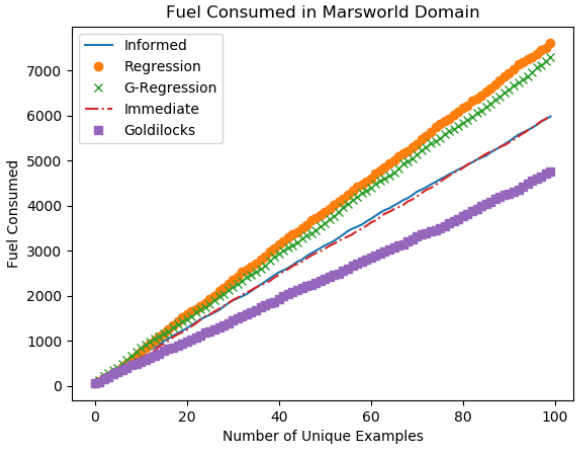

Fig. 4. Experimental results for our Marsworld Domain

after each time step). This damage may also cause the agent to lose beacon fuel. For testing, we ran 200 trials, each trial placed the rover and beacons randomly on the grid. During the trials we measured total fuel consumption as well as whether or not an execution failed. A failure means the preconditions of some action were not satisfied when it was to be executed (e.g., when the agent attempts to execute a move but it doesn't have enough fuel).

Results for Marsworld. In Figure 4, we can see that Regression and Goal Regression Expectations consumed the most fuel, with the Informed and Immediate performing basically equally. Goldilocks Expectations outperforms all of the rest. This occurs because Regression and Goal Regression Expectations are the only ones not noticing when the agent is damaged, causing increased fuel consumption. They only look at future preconditions, so they only realize the damage once it drains enough fuel so that the plan can no longer be completed. The other 3 expectation types identify increase consumption after 1 action, since they monitor effects of the actions. Goldilocks has the addition of noticing the mud from the regression part of its expectations, so it travels a more efficient route.

Immediate, Regression, Goal Regression and Goldilocks are able to ensure that the plan will be completed without failures, while 27% of trials failed for Informed Expectations. Informed fails because it will attempt to execute an action without it's preconditions being satisfied. All other expectation types check preconditions. Specifically, in this scenario, agents using Informed expectations will attempt to light a beacon after having lost beacon fuel, thus failing.

Camera Surveillance Definition. The agent in this domain is a camera with the ability to swivel 360° as well as zoom in and out. The goal of the agent in this domain is to keep a moving object in the environment in view, while as zoomed in as possible.

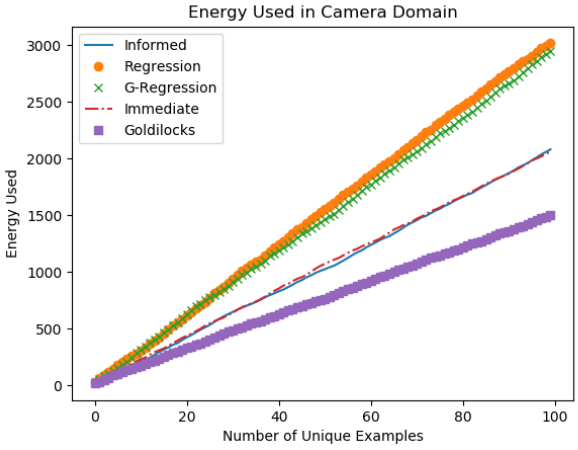

Fig. 5. Experimental results for our Camera Domain

Without a state satisfiable goal, we instead represented the goal as a reward function. Both zooming and turning the camera are durative actions in this domain, both consuming from the same energy resource. While executing its actions, the agent may suffer unexpected damage causing its actions to consume more energy. This damage is repairable if recognized. Since there is no satisfiable set of goals $\mathcal{G}$, goal regression and regression are equal.

Camera Surveillance Results. All expectation types were able to track the object, so the only difference between them is in energy consumption. In Figure 5, we can see that Regression and Goal Regression are very similar with the only differences coming from the random setup. We can also see these expectations used the most energy, while immediate and informed used a lower amount and Goldilocks used the lowest. This differential comes from the recognition of excess energy consumption, as well as efficient action taking. For example, both regression and goal regression only check the preconditions of actions. Therefore when an action consumes more energy than expected, the agent doesn't recognize it. This results in the agent continuing to take actions while consuming more energy per action, which results in more energy consumption per episode.

10 Conclusions

We see some commonalities from these two domains which are backed up by theory. Regression expectations may incur in a higher execution cost for the agent, but guarantee that the plan will succeed, illustrating Theorem 1. The increased cost comes from not monitoring the effects of actions previously executed. This allows for situations where the agent incurs excess cost without recognizing it. Informed and Immediate mitigate execution costs for the agent by checking the effects of previous actions. However by

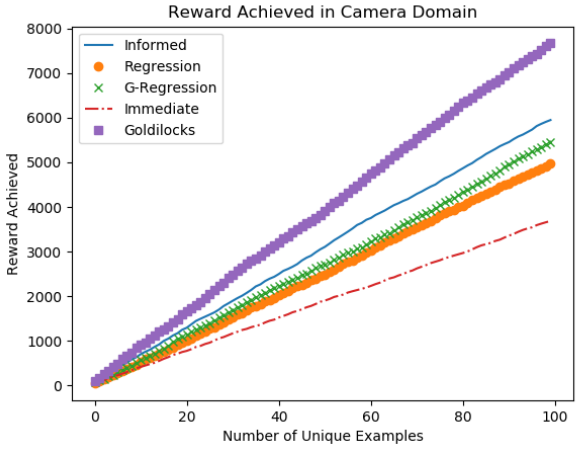

Fig. 6. Experimental results across Marsworld and Camera Domain

not checking the preconditions of future actions, the agent can fail to achieve its goals. Goldilocks is able to take the best from both and incur the smallest execution costs while guaranteeing the success of the plan. The agent checks its regressed conditions ensuring its goals are achieved, while monitoring the accumulated effects of its actions ensuring it doesn't incur in unexpected costs.

Our definition of expectations are not limited to GDA agents and could be used by other execution monitoring agents. In particular for future work, we will like to explore agent's expectations when the agent is part of a team achieving a common goal. This will require reasoning with social interaction aspects in addition to states' fluents.

References

1. D.W. Aha, AI Magazine (2018)
2. H. Muñoz-Avila, U. Jaidee, D. Aha, E. Carter, in *Case-Based Reasoning. Research and Development* (Springer, 2010), pp. 228–241
3. M. Molineaux, M. Klenk, D.W. Aha, in *AAAI* (2010)
4. D. Dannenhauer, H. Munoz-Avila, M.T. Cox, in *IJCAI* (2016), pp. 2493–2499
5. D. Dannenhauer, H. Munoz-Avila, in *IJCAI* (2015), pp. 2241–2247
6. D. Dannenhauer, Self monitoring goal driven autonomy agents. Ph.D. thesis, Lehigh University (2017)
7. N. Reifsnyder, H. Munoz-Avila, in *6th Goal Reasoning Workshop at IJCAI/FAIM-2018* (2018)
8. M. Molineaux, U. Kuter, M. Klenk, in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2* (International Foundation for Autonomous Agents and Multiagent Systems, 2012), pp. 989–996
9. J. Powell, M. Molineaux, D.W. Aha, in *FLAIRS Conference* (2011)

10. H. Munoz-Avila, D. Dannenhauer, N. Reifsnyder, in *Twentieth International Conference on Automated Planning and SchedulingProceedings of AAAI-19* (2019)
11. J.L. Pollock, Artificial Intelligence **106**(2), 267 (1998)
12. M.A. Wilson, J. McMahon, D.W. Aha, in *AI and Robotics: Papers from the AAAI Workshop* (2014)
13. N. Reifsnyder, H. Munoz-Avila, in *Eight Annual Conference on Cognitive Systems (ACS-2020)* (2020)
14. D. Wilkins, Computer Intelligence **1**, 33 (1985)
15. L. Birnbaum, G. Collins, M. Freed, B. Krulwich, in *AAAI*, vol. 90 (1990), vol. 90, pp. 318–323
16. M.T. Cox, AI magazine **28**(1), 32 (2007)
17. D.E. Wilkins, Computational Intelligence **1**(1), 33 (1985)
18. C. Fritz, S.A. McIlraith, in *ICAPS* (2007), pp. 144–151
19. E. Scala, P. Haslum, S. Thiébaux, M. Ramirez, in *Proceedings of the Twenty-second European Conference on Artificial Intelligence* (IOS Press, 2016), pp. 655–663
20. D. Nau. Pyhop, version 1.2.2 a simple htn planning system written in python. `https://bitbucket.org/dananau/pyhop` (2013). Accessed: 2019-01-30
21. M. Molineaux, D.W. Aha, in *AAAI* (2014), pp. 395–401