# Monitoring Numeric Expectations in Goal Reasoning Agents

**Noah Reifsnyder and Héctor Muñoz-Avila**
**Lehigh University**
**Bethlehem, PA 18015**
**{ndr217, hem4}@lehigh.edu**

### Abstract

One of the crucial capabilities for robust agency is self-assessment, namely the capability of the agent to compute its own boundaries. Goal reasoning agents do this by computing so called expectations: constructs defining the boundaries of their courses of action as a function of the plan, the goals achieved by that plan when available, the initial state, the action model and the last action executed. In this paper we introduce four forms of expectations when the agent reasons with numeric fluents and present an empirical evaluation highlighting their trade offs.

## Introduction

Over the past years there has been an increasing interest in goal reasoning agents; agents that may change their goals over time as a result of changes in the environment and/or changes in the user's requirements (Aha 2018). One of the main motivations for goal reasoning is the robust intelligence problem, where agents exhibit the following capabilities (Tianfield and Unland 2004):

- Are aware of their own limits or boundaries (self-assessment) (Sloman and Logan 1999).
- Recognize when they have stepped out of those boundaries (self-diagnosis) (Horling, Benyo, and Lesser 2001).
- Act to bring themselves back into their boundaries (self-correction) (Lucas 1961).

In this paper we focus on the **self-assessment** problem of goal reasoning agents. Our worked is motivated by the premise that it is infeasible for the agent to plan ahead for every possible contingency that it may encounter when executing a course of action. On the other hand, it is also unfeasible to replan to every possible contingency. In the words of Ghallab et al (2014) *"Current approaches in the planning literature either tend to foresee all possible events and situations, which is unpractical in realistic complex domains, or they tend to replan any time something unexpected occurs, which is hard to do in practice at run-time"*.

In domains with numeric fluents, it has been observed that it is contrived to assume after performing each action, each fluent will be expected to have an exact value (Scala et al. 2016). For instance when traveling between two location that a precise amount of gasoline will be consumed by the vehicle. Even under "usual" conditions, factors such as traffic accidents may cause delays. Authors have observed that a more robust way to handle numeric fluents is to use intervals or margins of error (Moore, Kearfott, and Cloud 2009). This reduces replanning since the agent can plan accounting for variations avoiding the need for repeated replanning. Nevertheless, the agent may encounter conditions under which fluents may take values outside predefined ranges.

Numeric fluents can be altered by actions through functions instead of by simple assignment of values (Hoffmann 2003; Coles et al. 2010). This presents a challenge to the concept of expectations for goal reasoning agents. Applying actions change symbolic fluents in the usual way (e.g., using add and delete lists) and change numeric fluents, representing a numeric value $v$, to a new value $f(v)$ as indicated by the actions' effects (Gerevini, Saetti, and Serina 2008). For example, for a *navigate* action, if $v = (energy\ ?x)$, then $f(v) = v - (t * r)$, where $t$ = *(travel-time ?y ?z)* and $r$ =*(use-rate ?x)*. The starting state, $s_0$ includes numeric fluents such as the fuel level of *rover21*, travel time between locations, and symbolic fluents such as the starting location of *rover21*. Frequently, for planning purposes these functions are assumed to be monotonic (Edelkamp 2003; Hoffmann 2003; Bajada, Fox, and Long 2015) although some paradigms drop this assumption (Scala et al. 2016).

In this paper we reexamine a taxonomy that encompasses expectations as computed by goal reasoning systems for symbolic fluents and extend that taxonomy for numeric fluents and their margins of error (Munoz-Avila, Dannenhauer, and Reifsnyder 2019). The taxonomy's starting point is the division of the plan $\pi = a_1..a_n$ generated between two parts $\pi_{prefix} = a_1..a_i$, the actions in $\pi$ already executed and $\pi_{suffix} = a_{i+1}..a_n$, the actions in $\pi$ to execute. We redefine the four forms of expectations from the goal reasoning literature for the numeric case: immediate, which checks the effects of the last action, $a_i$ executed; goal-regression, which computes the conditions needed to execute $\pi_{suffix}$; informed, which accumulated the effects from actions in $\pi_{prefix}$; and Goldilocks, which combines informed and regression expectations.

Up until now, the bulk of the research on goal reasoning has focused on domains where actions have symbolic fluents. The only exceptions we know are (Weber, Mateas, and Jhala 2012) and (Wilson, McMahon, and Aha 2014); these use immediate and informed expectations respectively.

The following are the main contributions of this paper:

- We extend the taxonomy of expectations in (Munoz-Avila, Dannenhauer, and Reifsnyder 2019) for plans with numeric fluents and margins of error.
- We analyze the tradeoffs between those expectations.

| |
|---|
| (:operator move_north<br>:parameters ?r<br>:condition not-within(at-y(?r), [0, 1]),<br>within( fuel(?r), [right(rate(?r)), $\infty$))<br>:effect<br>at-y(?r) = [$f_1$(?r), $f_2$(?r)],<br>fuel(?r) = [$f_3$(?r), $f_4$(?r)],<br>$f_1$(x) = left(at-y(x))-1,<br>$f_2$(x) = right(at-y(x))-1,<br>$f_3$(x) = left(fuel(x)) - right(rate(x)),<br>$f_4$(x) = right(fuel(x)) - left(rate(x)) |

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

| |
|---|
| (:Initial State<br>{fuel : {r23: [10,10]}}<br>{at-y : {r23: [2, 2], Beacon1: [0,0]}}<br>{at-x : {r23: [0,0], Beacon1: [2,2]}}<br>{lit : {Beacon1: [0, 0]}}<br>{rate : {r23: [.9, 1.1]}}<br>:Actions<br>move_north, move_south, move_east, move_west, light_beacon<br>:Goals<br>{lit : {Beacon1: [1, 1]}}<br>:Plan $\pi$<br>move_north, move_north, move_east, move_east, light_beacon |

Table 2: Planning problem and a solution plan

- We perform experiments on a goal reasoning system with the four forms of expectations.

Our work doesn't make any assumptions of how the plan $\pi$ was generated.

## States, Operators and Plans with Numeric Fluents

A state is a collection of variables that can be either symbolic or numeric. In this paper we will focus on actions with numeric fluents for simplicity of the exposition. A state is a mapping $s : V \rightarrow \mathbb{I}$ from a collection of variables $V$ to a collection of intervals $\mathbb{I}$. For instance, the variable *at-y(r23)* returns the y-coordinate as a confidence interval *[left(at-y(r23)),right(at-y(r23))]* for rover *r23*.

As exemplified in Table 1, an operator is a 4-tuple *o=(name parameters precondition effect)*. The parameters are a collection of free variables. We call free variables to variables that are not variables in the state. Free variables are used to facilitate writing operators. We denote free variables by using "?". For instance *?x* denotes the free variable x. The precondition is a set of numeric fluents and interval constraints where the preconditions can be represented as a partial mapping $pre : V \nrightarrow \mathbb{I}$. A fluent $nf$ is either represented as a closed interval $[left(nf), right(nf)]$, with $left(nf)$ and $right(nf)$ numbers and $left(nf) \leq right(nf)$; or if either side is unbounded, that bound is represented as an open interval bound on $\infty$ (or $-\infty$). Interval constraints are of the form *(within nf interval)* where $nf$ is a fluent. Within checks if $left(nf) \geq left(interval)$ and $right(nf) \leq right(interval)$. For instance, in the operator shown in Table 1 we are checking if the interval for the numeric fluent *fuel(?r)* is within the interval left bounded by $right(rate(?r))$ and right bounded by $\infty$. The effects indicate changes in value to the state variable. These are represented as function tuples $\mathbb{F} = \{(f_1, f_2) | f_1 \text{ and } f_2 \text{ are functions}\}$. The effects can be represented as the partial mapping $eff : V \nrightarrow \mathbb{F}$, where each variable $v \in V_{eff}$ therefore has the function tuple $(f_1^v, f_2^v)$.

An action takes a variable $v \in V$ and assigns $v$ value $[f_1(left(v)), f_2(right(v))]$, where $(f_1, f_2) \in \mathbb{F}$ and $f_1(x) <= f_2(x)$ for all $x$. For example, *fuel(?r)* represents the fuel level of vehicle $r$. Table 1 shows an example of an operator that uses *fuel(?r)*. The action $move\_north$ takes the variable $fuel(?r)$ and alters it with the following functions $fuel(?r) = [left(fuel(?r)) - right(rate(?r)), right(fuel(?r)) - left(rate(?r))]$. This denotes the usage of fuel by the action $move\_north$. The interval of rate(?r) represents amount of fuel we expect to be consumed by moving a tile. By subtracting the maximum of rate from the minimum of our current fuel level, we get the new minimum amount of fuel we expect to have after executing $move\_north$. By subtracting the minimum of the rate from the maximum of our current fuel level, we get the new maximum amount of fuel we expect to have. $move\_north$ also alters the variable at-y(?r). at-y(?r) is altered by the function tuple at-y$(?r) = (left($at-y$(?r)) - 1, right($at-y$(?r)) - 1)$. This denotes a change in y-position of the rover of -1 (north). So all together, the action $move\_north$ consumes the interval of rate(?r) amount of fuel while moving -1 in the y-coordinate plane.

## Two Basic Operations

We introduce two basic operations $\oplus_S$ and $\ominus_P$, which are used to define precisely the different forms of expectations.

We define $D = A \oplus_S B$, where $A$ are some variables, $S$ is the current state and $B$ are the effects of some action. More generally, for any partial functions $A$ and $B$ and and any function $S$ with $A : V \nrightarrow \mathbb{I}$, $S : V \rightarrow \mathbb{I}$, and $B : V \nrightarrow \mathbb{F}$, $A \oplus_S B$ is a partial mapping $D : V \nrightarrow \mathbb{I}$ defined as follows:

1. if $v \in V_A \cap V_B$ and $B(v) = (f_1, f_2)$, then $D(v) = [f_1(left(A(v))), f_2(right(A(v)))]$.
2. if $v \in V_A - V_B$ then $D(v) = A(v)$.
3. if $v \in V_B - V_A$ where $B(v) = (f_1, f_2)$, then $D(v) = [f_1(left(S(v))), f_2(right(S(v)))]$.
4. for all other variables $D$ is undefined (i.e., $V_D = V_A \cup V_B$)

Informally, $A \oplus_S B$ applies the function tuple in $B$ either to $A$ when the variable $v$ is defined in $A$ and $B$ (Case 1), or to $S$ when $v$ is defined in $B$ but not $A$ (Case 3). 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:

- $A = \{a : [2, 3]\}$
- $S = \{a : [2, 3], b : [7, 7], c : [8, 9], d : [6, 6]\}$

- $B = \{a : [x-2, x-1], b : [x+1, x+2], d : [x \times 2, x \times 3]\}$
- Then $D = A \oplus_S B = \{a : [0, 2], b : [8, 9], d : [12, 18]\}$.

In the resulting partial function $D(a) = [0, 2]$ is obtained by evaluating the functions tuple $B(a) = [x-2, x-1]$ on the interval $A(a) = [2, 3]$ (Case 1); $D(b) = [9, 11]$ is obtained by evaluating the functions tuple $B(b) = [x + 1, x + 2]$ on the value of $S(b) = [7, 7]$ (Case 3); and $D(d) = [12, 18]$ is obtained by evaluating the functions tuple $B(d) = [x \times 2, x \times 3]$ on the value of $S(d) = [6, 6]$ (Case 3).

We define $D = A \ominus_P B$, where $A$ are some variables, $P$ are the preconditions from some action and $B$ are the effects of the action. More generally, let $A : V \nrightarrow \mathbb{I}$, $P : V \nrightarrow \mathbb{I}$, and $B : V \nrightarrow \mathbb{F}$, we define $A \ominus_P B$ as a partial mapping $D : V \nrightarrow \mathbb{I}$ with:

1. if $v \in V_A - V_B$ then $D(v) = A(v)$.
2. if $v \in (V_A \cap V_B)$ and $B(v) = (f_1, f_2)$, then $D(v) = [f_1^{-1}(left(A(v))), f_2^{-1}(right(A(v)))]$.
3. if $v \in V_P - V_A$ then $D(v) = P(v)$
4. for all other variables $D$ is undefined (i.e., $V_D = V_A \cup V_P$)

Informally, $A \ominus_P B$ 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 applying the inverse of the functions tuple defined in $B(v)$ to the value of $A(v)$ (Case 2). If a variable $v$ is defined in $P$ but not in $A$, the new mapping takes the value of $P(v)$ (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:

- $A = \{a : [2, 3], b : [5, 6]\}$
- $P = \{c : [4, 4]\}$
- $B = \{b : [x + 1, x + 2]\}$
- Then $D = A \ominus_P B = \{a : [2, 3], b : [4, 4], c : [4, 4]\}$.

In the resulting function, $D(a) = [2, 3]$ because $a$ is defined in $A$ but not in $B$ (Case 1); $D(b) = [4, 4]$ because $A(b) = [5, 6]$ and $B(b) = [x + 1, x + 2]$, hence the inverse functions are $[x - 1, x - 2]$ (Case 2); and $D(c) = [4, 4]$ because $c$ is defined in P but not in A (Case 3).

## Immediate Expectations with Numeric Values

Immediate Expectations takes ideas from plan monitoring execution literature (see related work discussion). Formally, $X_{imm}(\pi, s_i, \emptyset) = imm_i$. Each $imm_i$ is generated as follows: $imm_0 = pre^{a_1}$. For all $i > 0, imm_i = (pre^{a_{i+1}} \ominus_{\{\}} eff_{s_i}^{a_i}) \oplus_{s_{i-1}} eff_{s_i}^{a_i}$. Informally, agents using immediate expectations check that the effects of the previous action $a_i$ and the preconditions of the next action to execute $a_{i+1}$ hold in the observed state $s_i$ (Cox 2007).

### Example:

In the plan $\pi$ in Table 2, assume we have completed $a_1$ = move-north and we are about to execute $a_2$ = move-north, then the Immediate Expectations, $imm_1 = (pre^{move\_north} \ominus_{\{\}} eff_{s_1}^{move\_north}) \oplus_{s_0} eff_{s_1}^{move\_north}$ with:

- $pre^{move\_north} = (\{fuel : \{r23 : [1.1, \infty]\}\}$
- $eff_{s_1}^{move\_north} = \{at\text{-}y : \{r23 : [x - 1, x - 1]\}, fuel : \{r23 : [x - 1.1, x - .9]\}\}$
- Then $(pre^{move\_north} \ominus_{\{\}} eff_{s_1}^{move\_north}) = \{fuel : \{r23 : [2.2, \infty]\}\}$
- and $imm_1 = (pre^{move\_north} \ominus_{\{\}} eff_{s_1}^{move\_north}) \oplus_{s_0} eff_{s_1}^{move\_north} = \{at\text{-}y : \{r23 : [1, 1]\}, fuel : \{r23 : [1.1, \infty]\}\}$

Since $fuel(r23)$ is a variable in common (Case 2 of the $\ominus_{\{\}}$ operation), we apply the inverse of $fuel : \{r23 : [x - 1.1, x-.9]\}$ which is $fuel : \{r23 : [x+1.1, x+.9]\}$ to $fuel : \{r23 : [1.1, \infty]\}$. The resulting value for $fuel(r23) = [(x + 1.1)(1.1), (x + .9)(\infty)) = [2.2, \infty)$

Finally, $X_{imm}(\pi, s_1, \emptyset) = \{fuel : \{r23 : [0, \infty]\}\} \oplus_{s_0} eff_{s_1}^{move\_north} = \{at\text{-}y : \{r23 : [1, 1]\}, fuel : \{r23 : [1.1, \infty]\}\}$ because $fuel(r23)$ is a common variable on the left and right side of $\oplus_{s_0}$ (Case 1 of the $\oplus_{s_0}$ operation) and *at-y(r23)* is a common variable in $s_0$ and $eff_{s_1}^{move\_north}$ (Case 3 of the $\oplus_{s_0}$ operation). Thus $fuel(r23) = [((x - 1.1)(2.2), (x - .9)(\infty)) = [1.1, \infty)$, and $(at\text{-}y(r23) = [(x-1)(2), (x-1)(2)] = [1, 1]$. This expectation set means that we expect to have at least 1.1 units of fuel and to be at y=1 on the coordinate frame (any x coordinate is fine)

## Informed Expectations with Numeric Values

Informed Expectations continuously build on effects from all previous actions executed so far in $\pi$. Informally, it compounds functions tuples $[left(v"), right(v")] = [f_1(\dots(f_1'(left(v)))\dots), f_2(\dots(f_2'(right(v)))\dots)]$, where $(f_1', f_2')$ are the effects for $v$ of the first action in the trace and $(f_1, f_2)$ are the effects for $v$ of the last action executed. If $v$ is not changed in some action $a$" then we assume $f_1"(x) = f_2"(x) = x$ (i.e., the identity function). Formally, $X_{inf}(\pi, s_i, \emptyset) = inf_i$. Each $inf_i$ is generated as follows: $inf_0 = \emptyset$. That is, before the first action is executed, we have no accumulated effects. For $i > 0$, $inf_i$ is defined recursively as follows: $inf_i = inf_{i-1} \oplus_{s_{i-1}} eff_{s_i}^{a_i}$. Agents using Informed Expectations check that the compounded effects are valid in the environment.

### Example:

If we have just completed action $a_2$ of the plan trace $\pi$ (the second instance of $move\_north$), we calculate the Informed Expectations $X_{inf}(\pi, s_2, \emptyset) = inf_2$ as follows from the initial state. We have:

- $inf_1 = \{at\text{-}y : \{r23 : [1, 1]\}, fuel : \{r23 : [8.9, 9.1]\}\}$
- $inf_2$ compounds $inf_1$ with the effects from $a_2$, the second $move\_north$:
  - $eff_{s_2}^{move\_north} = \{at\text{-}y : \{r23 : [x-1, x-1]\}, fuel : \{r23 : [x - 1.1, x - .9]\}\}$
- Thus, $inf_2 = inf_1 \oplus_{s_1} eff_{s_2}^{move\_north} = \{at\text{-}y : \{r23 : [0, 0]\}, fuel : \{r23 : [7.8, 8.2]\}\}$

For computing *at-y(r23)* we compute $[(x - 1)(1), (x - 1)(1)] = [0, 0]$ and for *fuel(r23)* we compute $[(x - 1.1)(8.9), (x - .9)(9.1)] = [7.8, 8.2]$. This expectation set

means that we expect to have between 7.8 and 8.2 units of fuel and to be at y=0 on the coordinate frame (any x coordinate is fine)

## Regression Expectations with Numeric Values

Regression Expectations continuously build on the cumulative values of the regressed conditions from all actions yet to be executed in $\pi$. Informally, it compounds functions tuples $[left(v'), right(v')] = [f_1^{-1}(\dots(f_{1'}^{-1}(left(v)))\dots), f_2^{-1}(\dots(f_{2'}^{-1}(right(v)))\dots)]$, where $(f_{1'}^{-1}, f_{2'}^{-1})$ are the inverse of the effects for $v$ of the last action in the trace and $(f_1^{-1}, f_2^{-1})$ are the inverse of the effects for $v$ of the next action to be executed. Formally, $X_{regress}(\pi, s_i, \mathcal{G}) = reg_i$. Each $reg_i$ is generated as follows: $reg_n = \mathcal{G}$. That is, when in the last state, the agent expects $\mathcal{G}$ to hold (when the goals are unknown, $\mathcal{G}$equals the empty set $\{\}$). For $i < n$, $reg_i$ is defined recursively as follows: $reg_i = reg_{i+1} \ominus_{pre^{a_{i+1}}} eff_{s_{i+1}}^{a_{i+1}}$. Informally, the agent checks the needed preconditions to execute the remaining of the plan $a_{i+1} \dots a_n$ while having the goals $\mathcal{G}$ satisfied in $s_n$.

### Example:

If we have just completed action $a_3$ of the plan trace $\pi$ in Table 2 (the first instance of $move\_east$), we calculate the Regression Expectations $reg_3 = X_{regress}(\pi, s_3, \mathcal{G})$ as follows (The preconditions and effects for $move\_east$ and $light\_beacon$ have not been shown before):

- $reg_5 = \mathcal{G} = \{lit : \{Beacon1 : [1,1]\}\}$ i.e. the Goals.
- $reg_4 = \{at{-}y : \{r23 : [0,0]\}, at{-}x : \{r23 : [2,2\}, lit : \{Beacon1 : [0,0\}\}$ which are the preconditions for $a_5, light\_beacon$.
- $reg_3 = reg_4 \ominus_{pre^{a_4}} eff_{s_4}^{a_4}$, where:
  - $pre^{a_4} = \{at{-}x : \{r23 : [0,1]\}, fuel : \{r23 : [1.1, \infty)\}\}$
  - $eff_{s_4}^{a_4} = \{at{-}x : \{r23 : [x+1, x+1]\}, fuel : \{r23 : [x-1.1, x-.9]\}\}$
- Thus, $reg_3 = \{at{-}y : \{r23 : [0,0]\}, at{-}x : \{r23 : [1,1]\}, fuel : \{r23 : [1.1, \infty)\}, lit : \{Beacon1 : [0,0\}\}$

$reg_4$ follows from the effects of $light\_beacon$, which increase $lit(Beacon1)$ by $(x+1, x+1)$ so the inverse generates $lit(Beacon1) = [(x-1)(1), (x-1)(1)] = [0,0]$. Therefore, $reg_3 = reg_4 \ominus_{pre^{move\_east}} eff_{s_4}^{move\_east}$, where $a_4$ is the second instance of $move\_east$. Since $fuel(r23)$ is in $pre^{move\_east}$ and not in $reg_4$, in $reg_3$: $fuel(r23) = pre^{move\_east}(fuel(r23)) = [1.1, \infty)$. Since $at{-}y(r23)$ and $lit(Beacon1)$ are in $reg_4$ and not in $eff_{s_4}^{move\_east}$, they just carry over from $reg_4$ into $reg_3$. $at{-}x(r23)$ is in both, $reg_4$ and $eff_{s_4}^{move\_east}$, so we compute the inverse functions from $eff_{s_4}^{move\_east}$ to get in $reg_3$: $at{-}x(r23) = [(x-1)(2), (x-1)(2)] = [1,1]$. This expectation set means that we expect to have at least 1.1 units of fuel, to be at y=0 and x=1 on the coordinate frame, and for the beacon to not be lit.

## Goldilocks Expectations with Numeric Values

Goldilocks Expectations (Reifsnyder and Munoz-Avila 2018) combines Informed and Regression Expectations. They are computed by calculating informed expectations from the starting state (i.e., in the numeric case by compounding functions $v = f(\dots(f'(v'))\dots)$), then regressing off from the final states the informed expectations (i.e., in the numeric case by compounding inverse functions $v'' = f'^{-1}(\dots(f^{-1}(v))\dots)$). This would not work in the numeric case since it will regress to the exact same value (i.e., $v" = v'$). We present a slightly different definition for Goldilocks Expectations; it takes into account both Informed and Regression Expectations independently.

The reason for doing this is because Regression can detect when the agent will not achieve its goals, while Informed can detect that something is wrong in the execution of the plan. By looking at them independently, the agent can make decisions over trade-offs between achieving the goals and how the agent is achieving them. For example, using the navigation domain from Table 2, an agent might have a goal to end the plan trace $\pi$ with a range of fuel left, e.g., $\{fuel : \{r23 : [0, \infty)\}\}$. In this scenario, Regression Expectations will make sure the agent has enough fuel to finish $\pi$. While this is important, it is also important to realize that Informed Expectations keep track of accumulated effects of the the actions executed so far. Informed is monitoring here the fuel consumption of the agent, and making sure it remains within the bounds as inferred from the action model. If the agent drifts out of those bounds, there may be a flaw with the agent causing it to consume more fuel than projected. So the agent might still achieve the goals but consume more fuel than expected. Recognizing this expectation failure can allow the agent to trigger a discrepancy and avoid needlessly wasting fuel. By considering both of these expectations, the agent can detect a variety of possible failures at their onset beyond "just" achieving the goals.

Formally, we define Goldilocks Expectations as $X_{gold}(\pi, s_i, \mathcal{G}) = gold_i$, where $gold_i = (inf_i, reg_i)$. That is, for ever state $s_i$, $gold_i$ is the pair containing the Informed and Regression Expectations for that state. An agent using $X_{gold}(\pi, s_i, \mathcal{G})$ 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:

When the agent completes action $a_3$ of the plan trace $\pi$ in Table 2 (the first instance of $move\_east$), it calculates the Goldilocks Expectations $gold_3$ as follows. $gold_3 = (inf_3, reg_3)$, both of which we exemplified previously as:

- $inf_3 = (\{at{-}y : \{r23 : [0,0]\}, at{-}x : \{r23 : [1,1]\}, fuel : \{r23 : [6.7, 7.3]\}\}$
- $reg_3 = \{at{-}y : \{r23 : [0,0]\}, at{-}x : \{r23 : [1,1]\}, fuel : \{r23 : [1.1, \infty)\}, lit : \{Beacon1 : [0,0]\}\})$

There is a difference in the two expectations over the expectations computed for variable $fuel(r23)$ . On the In-

formed side we expect between 6.7 and 7.3 fuel units, but on the Regression side, we just expect to have more than 1.1 fuel units. If we violate the Informed side but not the Regression side, we know we can likely finish the plan, but it will indicate a larger than expected fuel consumption.

## Empirical Evaluation

In our experiments, we tested 4 different types of expectations across numeric extensions of 2 domains used in the goal reasoning literature. The 4 Expectation types we tested were Immediate, Informed, Regression, and Goldilocks Expectations. For planning purposes, we use the Pyhop HTN planner (Nau 2013), which handles numeric fluents. 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 generate a goal to re-plan from the current state.

**Marsworld Definition** The first domain we used is a variant on the domain Marsworld (Dannenhauer, Munoz-Avila, and Cox 2016; Dannenhauer and Munoz-Avila 2015), inspired by Mudsworld (Molineaux and Aha 2014). The agent has to navigate a 10x10 grid to turn on 3 randomly placed beacons. Each movement action drains some amount of the agents fuel, which is determined by a predetermined rate for the agent. The agent also has known error rate for consuming fuel. Lighting each beacon also requires fuel, and consumes fuel from a different reserve then from where the agent draws from for movement.

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 5% probability after each action is taken). It can also lose some of its beacon fuel with a 5% probability after each action as well. During our testing, we ran 100 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 met when it was to be executed.

**Results for Marsworld**. In Figure 1, we can see that Regression Expectations consumed the most fuel, with the 3 other expectation types performing basically equally. The reason for this, is the Regression Expectations are the only ones not noticing when the agent is damaged, causing increased fuel consumption. Regression only looks at future preconditions, so it only realizes the damage once it drains enough fuel so that it no longer has enough to finish its plan. The other 3 expectation types identify increase consumption after 1 action, since they monitor effects of the actions.

Figure 2 shows the error rates for each expectation type Immediate, 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 met. All other expectation types check preconditions. Specifically, in this scenario, agents using Informed expectations will attempt to light a beacon after having lost some beacon fuel, thus failing the action.

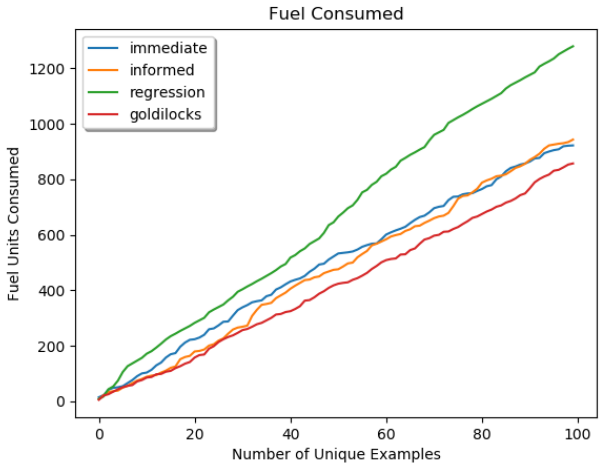


Figure 1: Accumulated Fuel Consumption in Marsworld across different types of expectations

**Blockscraft Definition**. The second domain we tested in is a variant of the Blockscraft domain (Dannenhauer, Munoz-Avila, and Cox 2016). In our variant, there are three towers of blocks; each block has a random mass. The mass of each block can only be estimated at planning time, so while the exact mass is unknown, a range for the mass is known and the exact mass is guaranteed to be within that range. We have an estimation of every blocks' mass. The agent can only access the top block in each tower, and can only know the exact mass of the block after collecting it. The task for the agent is to create a tower of blocks and the tower as a whole must be greater than a certain mass.

There is another actor in the environment who can take blocks both from the 3 towers our agent is using, as well as from the agent's own tower. There is an 8% chance after each action that the other actor will take a block out of the 3 center towers, and a 2% chance that the actor takes from the agent's tower. We ran 100 trials, where the mass of the blocks were randomized each trial. We measured total mass obtained by the agent during each trial, as well as if the trial failed or not. A failure happens either if the agent finishes the plan without enough mass in their tower, or an action's preconditions are not met when it is to be executed.

**Results for Blockscraft**. Figure 3 shows the total mass accumulated across 100 runs for an agent using each type of Expectations. Immediate, Informed, Regression, and Goldilocks all accumulated roughly the same amount of mass. They are all equal, because there isn't any discrepancies that alters the rate of obtaining mass, so the rates stay constant between all expectation types.

Figure 4 shows the failure rates. Only an agent using Goldilocks Expectations was able to complete the plan and have the goals fulfilled. Immediate failed 65% of its trials, while Regression failed 43% of its trials. These failures occurred when the other actor took blocks out of the agent's tower. Effects of actions are not monitored for those expectations, so the agent is not monitoring total mass of its tower.

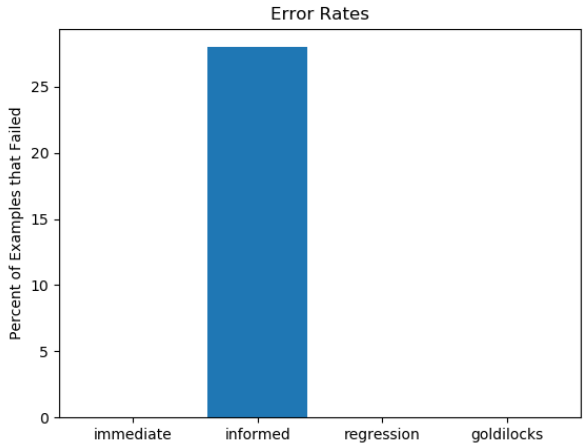


Figure 2: Number of failures in 100 executions in Marsworld across different types of expectations

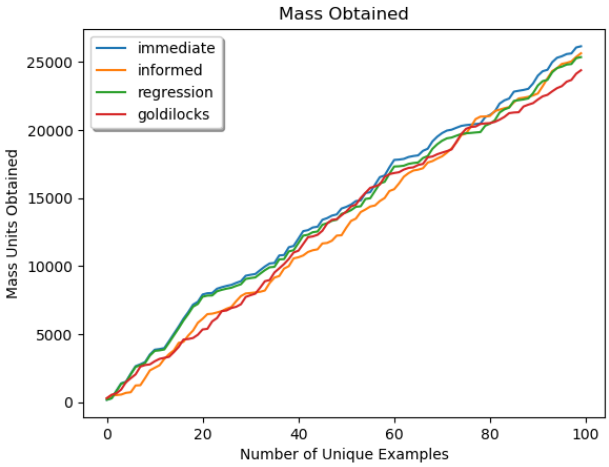


Figure 3: Accumulated Mass Obtained in the Blocks World Domain across different types of expectations

Informed failed 88% of its trials. This occurred due to the other actor removing blocks from the 3 central towers that the agent had planned to obtained. The action then to collect that block failed because the block didn't exist in the tower any more. Goldilocks had 0 failures across the 100 trials. This is because it checked preconditions due to the Regression side of the expectations, while also monitoring the mass so far obtained by the agent from the Informed side of the expectations. Combined, an agent using Goldilocks Expectations caught all discrepancies.

## Related Work

(Scala 2013) proposed using kernel methods to compute the necessary numerical conditions $K^i$ needed to complete the rest of the plan $a_i \dots a_n$, akin to our regression expectations. While not defined that way, $K^i(v) = f_i^{-1}(\dots(f_n^{-1}(v))\dots)$, where $v$ is a goal condition and $f_j^{-1}$ is the inverse function for $v$ in $a_j$ (with $i \leq j \leq n$). (Scala and Torasso 2014) expands this to distances around the values of $v$ playing a similar role as our error intervals. Our work differs in some important differences: by explicitly using inverse functions we provide a concrete way to compute these kernels. More importantly, we introduce two forms of expectations: informed and Goldilocks. Informed expectations are needed when goals are not known. Informed expectations can provide needed information missing from goal regression calculations. For example, in a scenario where we have resource consumption, if we have an action that consumes an amount of resource, we would likely have some goal to have $> 0$ amount of the resource (or more than however much the action consumes). If there are multiple occurrences of this action, and they end up consuming a larger amount of resource, informed will allow us to (1) detect the discrepancy after the first action, instead of after however many it takes to deplete the resource and (2) conserve more of the resource. Crucially, Goldilocks allows the detection of

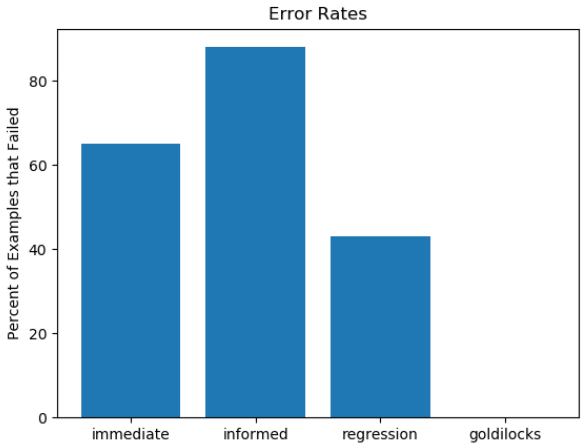


Figure 4: Number of failures in 100 executions in Blocks World Domain across different types of expectations

deviations in the values of numerical values that are pivotal in finishing the provided plan with the expected outcomes, even when the goals are unknown.

We know of two systems using numeric fluents for goals reasoning. (Weber, Mateas, and Jhala 2012) represents quantities as numeric fluents for a goal reasoning agent playing an adversarial real-time computer game. For instance, an action to *produce 10 archers*, will have as expectation that 10 archers are produced. After executing the action, if the number of archers is $num$ with $num < 10$, then a discrepancy is detected. The agent will formulate a new goal to produce $10 - num$ archers. In the context of the taxonomy we presented, this agent maintains immediate expectations. Furthermore, no margins of error are maintained as the agents expectations are exact natural numbers.

(Wilson, McMahon, and Aha 2014) uses what we call informed expectations. They project forward the expected

numerical values within intervals and detect discrepancies when the values are outside of these projected intervals. Our experiments show how informed expectations can incur into the highest number of errors because they don't regress conditions on the goals. This leads to a high volume of failures when they attempt to be executed.

Plan monitoring execution systems annotate the plan with conditions necessary for the plan's execution to be valid (Fikes, Hart, and Nilsson 1972). While not using "expectations" as a term, (Ambros-Ingerson and Steel 1988) checks for the causal links, triples *(effect, fluent, precondition)* are met when the action having the precondition is to be executed. In our parlance, this is subsumed by immediate expectations. Plan monitoring execution have also been used to monitor optimality. For instance, (Fritz and McIlraith 2007) uses goal regression to define necessary conditions to guarantee the optimal execution of the plan. These works use symbolic fluents.

## Conclusions

We introduce 4 forms of Expectations over numerical fluents: Immediate, Informed, Regression, and Goldilocks. In our empirical evaluation, only agents using Goldilocks solved problems without failures across all domains; Goldilocks projects forward all changes the agent makes to the state, as well as making sure the agent is on track to meet the goals. The agent using Regression was shown to be inefficient over the fuel consumption in one of the domains. The reason is that Regression only checks if the agent is on track to satisfy the goals but makes no consideration of any other deviation from the action model.

Time series have been used to build statistical models of "normal" or expected readings for numerical values and in doing so detect outliers to predict malfunctions (Tsay 1988). In this work, we assume the "normal" ranges has been given in the effects of the actions. However, in future work, such models could be used in situations when we expect the effects of the actions to change over time.

**Acknowledgements.** This research was supported by ONR under grants N00014-18-1-2009 and N68335-18-C-4027.

## References

Aha, D. W. 2018. Goal reasoning: foundations emerging applications and prospects. *AI Magazine*.

Ambros-Ingerson, J. A., and Steel, S. 1988. Integrating planning, execution and monitoring. In *AAAI*, volume 88, 21–26.

Bajada, J.; Fox, M.; and Long, D. 2015. Temporal planning with semantic attachment of non-linear monotonic continuous behaviours. In *IJCAI*, 1523–1529.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Twentieth International Conference on Automated Planning and Scheduling*.

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

Dannenhauer, D., and Munoz-Avila, H. 2015. Raising expectations in gda agents acting in dynamic environments. In *IJCAI*, 2241–2247.

Dannenhauer, D.; Munoz-Avila, H.; and Cox, M. T. 2016. Informed expectations to guide gda agents in partially observable environments. In *IJCAI*, 2493–2499.

Edelkamp, S. 2003. Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Intelligence Research* 20:195–238.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Some new directions in robot problem solving. *Machine Intelligence* 7:405–430.

Fritz, C., and McIlraith, S. A. 2007. Monitoring plan optimality during execution. In *ICAPS*, 144–151.

Gerevini, A. E.; Saetti, A.; and Serina, I. 2008. An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artificial Intelligence* 172(8-9):899–944.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research (JAIR)* 20:291–341.

Horling, B.; Benyo, B.; and Lesser, V. 2001. Using self-diagnosis to adapt organizational structures. In *Proceedings of the fifth international conference on Autonomous agents*, 529–536. ACM.

Lucas, J. R. 1961. Minds, machines and gödel. *Philosophy* 36(137):112–127.

Molineaux, M., and Aha, D. W. 2014. Learning unknown event models. In *AAAI*, 395–401.

Moore, R. E.; Kearfott, R. B.; and Cloud, M. J. 2009. *Introduction to interval analysis*, volume 110. Siam.

Munoz-Avila, H.; Dannenhauer, D.; and Reifsnyder, N. 2019. Is everything going according to plan? - expectations in goal reasoning agents. In *Proceedings of AAAI-19*.

Nau, D. 2013. Pyhop, version 1.2.2 a simple htn planning system written in python. `https://bitbucket.org/dananau/pyhop`. Accessed: 2019-01-30.

Reifsnyder, N., and Munoz-Avila, H. 2018. Goal reasoning with goldilocks and regression expectations in nondeterministic domains. In *6th Goal Reasoning Workshop at IJCAI/FAIM-2018*.

Scala, E., and Torasso, P. 2014. Proactive and reactive reconfiguration for the robust execution of multi modality plans.

Scala, E.; Haslum, P.; Thiébaux, S.; and Ramirez, M. 2016. Interval-based relaxation for general numeric planning. In *Proceedings of the Twenty-second European Conference on Artificial Intelligence*, 655–663. IOS Press.

Scala, E. 2013. Numeric kernel for reasoning about plans involving numeric fluents. In *Congress of the Italian Association for Artificial Intelligence*, 263–275. Springer.

Sloman, A., and Logan, B. 1999. Building cognitively rich agents. *Communications of the ACM* 42(3):71–72.

Tianfield, H., and Unland, R. 2004. Towards autonomic

computing systems. *Engineering Applications of Artificial Intelligence* 17(7):689–699.

Tsay, R. S. 1988. Outliers, level shifts, and variance changes in time series. *Journal of forecasting* 7(1):1–20.

Weber, B. G.; Mateas, M.; and Jhala, A. 2012. Learning from demonstration for goal-driven autonomy. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Wilson, M. A.; McMahon, J.; and Aha, D. W. 2014. Bounded expectations for discrepancy detection in goal-driven autonomy. In *AI and Robotics: Papers from the AAAI Workshop*.