An Argumentation-Based Interpreter for Golog Programs

Michelle L. Blom and Adrian R. Pearce
NICTA Victoria Research Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne
{mlblom, adrian}@csse.unimelb.edu.au

Abstract

This paper presents an argumentation-based interpreter for Golog programs. Traditional Golog interpreters are not designed to find the most preferred executions of a program from the perspective of an agent. Existing techniques developed to discover these executions are limited in terms of how the preferences of an agent can be expressed, and the variety of preference types that can be used to guide search for a solution. The presented work combines the use of argumentation to compare executions relative to a set of general comparison principles, and the theory behind best first search to reduce the cost of the search process. To the best of our knowledge this is the first work to integrate argumentation and the interpretation of Golog programs, and to use argumentation as a tool for best first search.

1 Introduction

Golog [Levesque *et al.*, 1997] is a logic programming language for agents designed to represent complex actions and procedures in the situation calculus. Each execution of a Golog program represents a way of achieving a particular goal. A traditional Golog interpreter is designed to find the possible executions of a program, not to identify those that are the most preferred.

Several approaches have been developed to achieve this task, and extend Golog with preferences. DTGolog – a decision theoretic extension of Golog – maximises utility in the interpretation of Golog programs [Boutilier *et al.*, 2000]. In [Fritz and McIlraith, 2006], DTGolog is extended with the incorporation of qualitative preferences in the form of temporal logic constraints of varying priority. In [Sohrabi *et al.*, 2006], best first search is applied in the interpretation of Golog programs to discover a web service composition that is the most preferred given temporal logic preference formulae.

The variety of preference types that can be expressed in these existing approaches is limited. The utilities of [Boutilier *et al.*, 2000] and the qualitative preferences of [Fritz and McIlraith, 2006; Sohrabi *et al.*, 2006] do not express all the ways in which executions of a program can be compared. These approaches, for example, cannot capture comparative preference – derived from the relationships that exist between one choice and another. Examples of such preferences include those based on: intangible, immeasurable factors such as taste and perception; decision heuristics encapsulating past decision-making experience; and lexicographic or semi-lexicographic [Tversky, 1969] comparison rules. In this paper, we present an argumentation-based Golog interpreter that aims to address this limitation, while maintaining the advantage of reduced search cost through best first search.

Argumentation techniques have gained widespread use in both agent and multi-agent systems as tools for the support of more human-like decision-making processes [Ouerdane *et al.*, 2007]. In this paper, we first develop an argumentation framework for decision-making (ADF) that constructs a strict preference profile (a *dominance graph*) over a set of executions $\mathcal{A}$ of a Golog program. This framework allows an agent to compare executions according to whichever principles it desires (denoted *comparison principles*), including those describing comparative preference. We do not assume that the resulting profile is transitive or acyclic, and as such we apply choice theory techniques to define the most preferred subset of $\mathcal{A}$ ($MPS(\mathcal{A})$). We apply a choice rule, such as the Schwartz [Schwartz, 1972] criterion, to a dominance graph over $\mathcal{A}$ to select these most preferred alternatives.

We then characterise the interpretation of a Golog program as best first search (using the transition semantics of [de Giacomo *et al.*, 2000]). At each stage in the search for a most preferred execution of a program δ, an instantiation of our ADF provides an estimate of which available transitions will lead to an execution in $MPS(\mathcal{A})$, where $\mathcal{A}$ denotes the executions of δ. The concept of *speculation* (a tool for predicting future opportunities in search) and a prioritised selection algorithm are used to select a transition to perform at each stage of the search process. To select a transition, an agent speculates on (estimates) the nature of the preferences that exist between complete executions of a program, based on the preferences that exist between partial executions at each stage in search.

Our argumentation-based best first search interpreter is shown to always find an execution in the most preferred subset of a program's execution set $\mathcal{A}$ (provided $\mathcal{A} \neq \emptyset$).

The remainder of this paper is structured as follows. In Section 2, we provide an introduction to the Golog programming language. In Section 3, we describe in more detail the existing preference-based Golog interpreters, their limitations, and our motivation for using argumentation-based best first search to address them. Our ADF, defining $MPS(\mathcal{A})$ for a set of program executions $\mathcal{A}$, is outlined in Section 4. In Section 5, we describe argumentation-based best first search.

2 Golog, and The Situation Calculus

Golog programs are described using axioms of the situation calculus, together with constructs to encode procedures, actions, iteration, and non-deterministic choice [Levesque *et al.*, 1997]. The situation calculus enables reasoning about *actions*, which change the state of the world, and *situations*, which describe a sequence of actions applied to an initial state. In the situation calculus, a collection of action precondition, successor state, foundational domain independent, and initial state axioms characterise the domain of an application.

The constructs present in the Golog programming language are described in [de Giacomo *et al.*, 2000; Levesque *et al.*, 1997] and are: a (primitive action); $\phi?$ (test that ϕ holds in the current situation); $\delta_1; \delta_2$ (execute program δ_1 then δ_2); $\delta_1 | \delta_2$ (choose to execute program δ_1 or δ_2); $\pi\ v.\delta$ (choice of arguments v in δ); δ^* (execute δ zero or more times); **if** ϕ **then** δ_1 **else** δ_2 (synchronised conditional); **while** ϕ **do** δ (synchronised loop); and **proc** $P(\vec{v})\delta$ **end** (procedure).

Given a domain theory, D, expressed in the situation calculus, a legal execution of a Golog program δ is a sequence of actions $\vec{a}$ such that: $D \models Do(\delta, s_o, do(\vec{a}, s_o))$, where $do(\vec{a}, s_o)$ is a legal terminating situation (execution) of δ when executed in the situation s_0[1]. Each pair (δ, s), where δ is a program to be executed in situation s, is called a configuration.

The transition semantics for the interpretation of Golog programs [de Giacomo *et al.*, 2000] (providing an implementation of Do) defines when an agent can transition from one configuration (δ, s) to another (δ', s') by performing one step in the program δ, resulting in the situation s' and the program δ' left to perform. For each form a Golog program may assume, δ_f, a *Trans* axiom is defined to determine which configurations can be transitioned to from (δ_f, s) by performing a single step of δ_f in situation s. A configuration (δ_f, s) is final (*Final*(δ_f, s)) if no steps remain to be performed in δ_f.

A selection of *Trans* and *Final* axioms are shown below[2]. Given a program of the form $\delta_f = a$ (where a is a primitive action), the configuration (δ_f, s) can transition if it is possible to perform action a in situation s ($Poss(a[s], s)$),

$$Trans(a, s, \delta', s') \equiv$$
$$Poss(a[s], s) \wedge \delta' = nil \wedge s' = do(a[s], s)$$

and is not final as the action a remains to be performed. For the sequence $\delta_f = \delta_1; \delta_2$ (where δ_1 is performed first and then δ_2), the configuration (δ_f, s) transitions as follows,

$$Trans(\delta_1; \delta_2, s, \delta', s') \equiv \exists\gamma.\delta' = (\gamma; \delta_2) \wedge$$
$$Trans(\delta_1, s, \gamma, s') \vee Final(\delta_1, s) \wedge Trans(\delta_2, s, \delta', s')$$

and is final if both δ_1 and δ_2 are final in s:

$$Final(\delta_1; \delta_2, s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s)$$

[1] do($\{a_1, a_2, \ldots, a_{n-1}, a_n\}$, s) is shorthand for do(a_n, do(a_{n-1}, ... do(a_2, do(a_1, s)))).

[2] A full account of the transition semantics can be found in [de Giacomo *et al.*, 2000].

3 Related Work

In rational choice theory, it is assumed that the preferences of a decision maker can be expressed as a utility function that assigns a value to each decision alternative. Utility maximisation produces a complete and transitive ranking from which a most preferred alternative can be selected.

Golog and Preferences Developed in this vein is DTGolog – a decision theoretic extension of Golog [Boutilier *et al.*, 2000]. DTGolog – a prominent approach for finding the most preferred executions of a Golog program – combines the theory of Markov decision processes (MDPs) with the Golog programming language. Solving an MDP involves finding an optimal policy – an optimal mapping between the states an agent might find itself in and the actions that should be performed in those states. DTGolog specifies an MDP with an action theory (expressed in the situation calculus) augmented with stochastic actions (actions with uncertain outcomes) and a reward function mapping numerical rewards to situations. A policy for a program δ is optimal if it maximises the expected reward or utility achieved over a given horizon (maximum number of actions performable), and the probability with which δ will be successfully executed.

The incorporation of qualitative preferences in the interpretation of Golog programs [Fritz and McIlraith, 2006; Sohrabi *et al.*, 2006] and planning in general [Son and Pontelli, 2004; Bienvenu *et al.*, 2006] is an advantage when quantitative preferences (such as utilities or rewards) are not available or do not capture all of the preferences of an agent.

Fritz and McIlraith extend DTGolog with qualitative preferences in [Fritz and McIlraith, 2006]. These preferences are expressed in terms of basic desire formulae (temporal logic constraints on situations) of varying priority. These formulae are compiled into a DTGolog program to be executed synchronously with the program under interpretation δ. This synchronous execution aims to find the most quantitatively preferred policies of δ amongst those that satisfy (if not all, at least some of) the qualitative constraints.

Sohrabi *et al* develop GologPref [Sohrabi *et al.*, 2006], a Golog interpreter designed to discover a most preferred web service composition using the best first search strategy of PPLAN [Bienvenu *et al.*, 2006]. In [Bienvenu *et al.*, 2006], user preferences are expressed in a language that extends $\mathcal{PP}$ [Son and Pontelli, 2004]. $\mathcal{PP}$ describes a hierarchy of preference constructs: basic desire formulae outlining temporal logic constraints on plan trajectories; atomic preferences specifying an ordering on basic desires according to their importance; and general preference formulae allowing the combination of atomic preferences with operators analogous to $\wedge$, $\vee$, and $\neg$. PPLAN introduces additional mechanisms of preference combination and assigns weights (denoting degrees of importance) to basic desires in atomic preference formulae. These weights are used to compute a value for each partial plan on a search frontier, where preferences not already violated are assumed to be satisfied at a future point on the trajectory of a partial plan. PPLAN selects the most promising partial plan for exploration at each stage in search.

In contrast with the work of [Sohrabi *et al.*, 2006], DTGolog based approaches [Boutilier *et al.*, 2000; Fritz and

McIlraith, 2006] consider the entire execution space in the search for one that is most preferred. This need arises, however, from their incorporation of stochastic actions. In this paper, we do not consider the use of stochastic actions.

Limitations of Existing Work In [Delgrande *et al.*, 2007], a distinction between absolute preference (describing the desirability of a particular state of affairs) and comparative preference (describing the desirability of one state of affairs relative to others) is made. The approaches described above support only absolute preference, and disallow those that imply the merit of one partial plan or execution is dependent on the alternatives it is being compared with.

In addition to the varieties of preference that can often only be described comparatively – such as preference over tastes and perception – preference over alternatives involving multiple attributes is often expressed in terms of comparative rules to reduce the cognitive load on a decision maker [Bar-Hillel and Margalit, 1988]. An example is the application of a majority rule over multiple attributes. In this scenario, one alternative is preferred to another if it is better on a majority of attributes. Approximation methods used to ascertain a preference for one alternative over another – such as the majority rule described above, and the cancelling out of dimensions with similar differences [Tversky, 1969] – are common in human decision-making [Tversky, 1969].

Argumentation and Best First Search Argumentation techniques provide a general means by which preference between alternatives can be specified or ascertained. In argumentation-based decision-making, preference amongst alternatives is *constructed* as a part of the decision-making process, and is based on the competing interests, motivations, values, or goals, of the decision maker. Given a set of program executions, reasons may be formed in support of preferences between its members based on the variety of principles an agent may use to compare them.

In this paper, we combine the advantages of argumentation as a tool for expressing complex preferences, and best first search – a technique applied in existing work to reduce the cost of search for a most preferred execution. In place of independently evaluating partial executions with respect to a set of absolute preferences, arguments supporting preferences between partial executions are used to guide the search.

4 Argumentation over Executions

In this section, we define an argumentation framework for decision-making (*ADF*) that constructs preference relationships between decision alternatives given a general set of principles by which they are compared. We apply this framework, in conjunction with choice theory, to find the most preferred subset of an execution set $\mathcal{A}$ for a Golog program δ.

We first provide the following definition of a *comparison principle*, and assume that an agent $a = \langle \mathbf{C}, \mathcal{I} \rangle$ is equipped with a finite set of such principles $\mathbf{C}$, and a (transitive, irreflexive, and asymmetric) partial ordering $\mathcal{I}$ over the principles in $\mathbf{C}$. Given $c_1, c_2 \in \mathbf{C}$, $\mathcal{I}(c_1, c_2)$ denotes that principle c_1 is more important to a than principle c_2.

Definition 1 *Given decision alternatives $d_1, d_2 \in \mathcal{D}$, a comparison principle is a relation $\mathcal{C}(d_1, d_2, \phi)$ where ϕ is a conditional expression involving d_1 and d_2. If $\phi(d_1, d_2)$ holds, we say that d_1 is preferred to d_2 ($\mathcal{P}(d_1, d_2)$).*

We now introduce a running example that will used throughout the remainder of this paper.

Example 1 (The Travelling Agent) *An agent a has the following program (denoted δ) outlining what to do to travel from point A to B to C. Agent a can drive or walk from A to B, and then take the train (preceded by eating cake) or skip (followed by eating fruit) from B to C.*

proc $travel(A, B, C)$
  $[drive(A, B) | walk(A, B)]$;
  $[[eat(cake); train(B, C)] \mid [skip(B, C); eat(fruit)]]$
end

We make the assumption that all actions are possible in any situation, with the exception of taking the train which is not possible in a situation that has involved driving. Given an initial situation s_0, $A =$ home (h), $B =$ university (u), $C =$ cinema (c), the program δ has three executions:

$$s_1 = do(\{drive(h, u),\ skip(u, c),\ eat(fruit)\},\ s_0),$$
$$s_2 = do(\{walk(h, u),\ eat(cake),\ train(u, c)\},\ s_0),$$
$$s_3 = do(\{walk(h, u),\ skip(u, c),\ eat(fruit)\},\ s_0).$$

Agent a has three principles $\mathcal{C}_i(s_j,\ s_k,\ \phi_i)$ by which it compares these executions. By the first comparison principle, agent a prefers one execution over another if it involves eating tastier food.

$$\phi_1(do(\vec{a_1}, s_0), do(\vec{a_2}, s_0)) \equiv eat(x) \in \vec{a_1} \wedge eat(y) \in \vec{a_2} \wedge x >_{taste} y$$

where $x >_{taste} y$ denotes that x tastes better than y. The second principle expresses that walking and skipping together is preferred to driving.

$$\phi_2(do(\vec{a_1}, s_0), do(\vec{a_2}, s_0)) \equiv walk(x_i, y_i) \in \vec{a_1} \wedge skip(x_j, y_j) \in \vec{a_1} \wedge drive(x_k, y_k) \in \vec{a_2}$$

By the third principle, agent a prefers executions involving less tiring activities.

$$\phi_3(do(\vec{a_1}, s_0), do(\vec{a_2}, s_0)) \equiv x \in \vec{a_1} \wedge y \in \vec{a_2} \wedge x <_{tiring} y$$

where $x <_{tiring} y$ denotes that x is less tiring than y.

There are several existing frameworks that have been developed to argumentatively determine the 'best' decisions in an alternative set based on the preferences of an agent (see [Ouerdane *et al.*, 2007] for a review). Each of these approaches is designed to induce a preference ordering over a set of available decisions, from which the best (most preferred) decisions can be inferred. This preference ordering is formed by analysing and comparing a set of arguments (*pro* and *con* each decision, for example) constructed in light of the positive and negative aspects of each decision.

Our aim is to provide an agent with the flexibility to compare executions with whichever principles it desires. Rather than construct arguments in support of and against individual decisions (*pros* and *cons*), comparison principles are instantiated in our ADF to form arguments in support of preference relationships *between* decisions (executions).

Each argument in our *ADF* represents a reason why one execution $s_i \in \mathcal{A}$ is preferred to another $s_j \in \mathcal{A}$ ($\mathcal{P}(s_i, s_j)$). Each argument takes the form $\langle J, \mathcal{P}(s_i, s_j)\rangle$ where s_i, $s_j \in \mathcal{A}$ and J is a justification for the truth of the conclusion $\mathcal{P}(s_i, s_j)$. Each justification J represents the instantiation of a comparison principle $c \in \mathbf{C}$ used by an agent to determine if one execution is preferred to another. An argument $a_1 = \langle J_1, \mathcal{P}(s_i, s_j)\rangle \in AR$ is attacked (rebutted) by all arguments with an opposing conclusion $a_2 = \langle J_2, \mathcal{P}(s_j, s_i)\rangle \in AR$ (denoted $attacks(a_2, a_1)$). An argument a_1 defeats an argument a_2 iff $attacks(a_1, a_2)$ holds and the principle that derives a_2 is not more important to the decision maker (agent) than the principle that derives a_1.

The structure of our *ADF* is based on the structure of an audience-specific value-based argumentation framework [Bench-Capon, 2003]. The conclusions of undefeated arguments (arguments not defeated by another argument) in an *ADF* represent a strict preference profile (a dominance graph) $\mathcal{P}$ over the available decisions (executions).

Definition 2 *An ADF is a 5-tuple:* $ADF = \langle$AR, attacks, ***C***, pri, $\mathcal{I}\rangle$ *where* AR *is a finite set of arguments,* attacks *is an irreflexive binary attack relation on* AR, ***C*** *is a finite set of comparison principles,* pri *maps each argument in* AR *to the principle in* ***C*** *it is derived from, and* $\mathcal{I} \subseteq \boldsymbol{C} \times \boldsymbol{C}$ *is a (transitive, irreflexive, and asymmetric) partial ordering of the principles in* ***C***. *An argument* a $\in$ AR *defeats an argument* b $\in$ AR *iff* attacks(a, b) *and* $\neg\mathcal{I}$(pri(b), pri(a)) *holds.*

We assume that an agent has its own choice rule (Ch) for selecting a most preferred subset of $\mathcal{A}$ ($MPS(\mathcal{A})$) based on the preferences that have been found to exist between its elements ($Ch(\mathcal{A}, \mathcal{P}) = MPS(\mathcal{A})$). As preference is based on a set of general comparison principles, we do not assume that $\mathcal{P}$ is transitive or acyclic. An agent may, for example, represent $Ch(\mathcal{A}, \mathcal{P})$ with the Schwartz set [Schwartz, 1972].

Definition 3 *Given a set of alternatives* $\mathcal{D}$*, and a dominance graph over* $\mathcal{D}$*,* $\mathcal{P}$*: (i) an alternative* $\gamma \in \mathcal{D}$ *dominates an alternative* $\beta \in \mathcal{D}$ *iff* $(\gamma, \beta) \in \mathcal{P}$*; (ii) a Schwartz set component of* $\mathcal{D}$ *is a subset* $S \subseteq \mathcal{D}$ *such that: (a) no alternative* $\gamma \in \mathcal{D}$ *where* $\gamma \notin S$ *dominates an alternative* $\beta \in S$*; and (b) no non-empty subset* $S' \subset S$ *satisfies property (a); and (iii) the Schwartz set* Sc *of* $\mathcal{D}$ *is the union of all Schwartz set components of* $\mathcal{D}$*.*

If a preference cycle exists between decision alternatives, the Schwartz set infers that they are each ‘at least as good as each other’ [Schwartz, 1972].

Example 2 (The Travelling Agent cont.) *For agent a:* fruit $>_{taste}$ cake, drive $<_{tiring}$ walk, *and* $\mathcal{I} = \{(\mathcal{C}_1, \mathcal{C}_2), (\mathcal{C}_1, \mathcal{C}_3), (\mathcal{C}_2, \mathcal{C}_3)\}$. *In an* ADF *defined over* $\mathcal{A} = \{s_1, s_2, s_3\}$*, a is able to instantiate its comparison principles to produce five arguments in support of preference relationships between elements of* $\mathcal{A}$ ($AR = \{A_i\}_{i=1,\ldots,5}$).

Argument 1, $A_1 = \langle \mathcal{C}_1, \mathcal{P}(s_1, s_2)\rangle$*, denotes that* s_1 *is preferred to* s_2 *as it involves eating tastier food. Similarly,* $A_2 = \langle \mathcal{C}_1, \mathcal{P}(s_3, s_2)\rangle$*. Argument 3,* $A_3 = \langle \mathcal{C}_2, \mathcal{P}(s_3, s_1)\rangle$*, denotes that* s_3 *is preferred to* s_1 *as* s_3 *involves walking and skipping while* s_1 *involves driving. Argument 4,* $A_4 = \langle \mathcal{C}_3, \mathcal{P}(s_1, s_2)\rangle$*, denotes that* s_1 *is preferred to* s_2 *as it is less tiring. Similarly,* $A_5 = \langle \mathcal{C}_3, \mathcal{P}(s_1, s_3)\rangle$*.*

By Definition 2, A_3 *and* A_5 *attack each other with* A_3 *defeating* A_5 *(as* $\neg\mathcal{I}(\mathcal{C}_3, \mathcal{C}_2)$*), and* $\mathcal{P} = \{(s_1, s_2), (s_3, s_2), (s_3, s_1)\}$*. Given the Schwartz set (Definition 3) as* Ch*, the most preferred subset of* $\mathcal{A}$*,* $MPS(\mathcal{A})$*, is* $Ch(\mathcal{A}, \mathcal{P}) = \{s_3\}$*.*

5 Argumentation-Based Best First Search

In this section, we use the transition semantics defined in [de Giacomo *et al.*, 2000] to characterise the interpretation of a Golog program as best first search. Given a program δ to interpret (and an initial situation s_0), the state-space of the problem is the set of configurations (δ_i, s_i) that can be transitioned to from (δ, s_0) by performing a sequence of actions in δ. Given a configuration (δ_i, s_i), a legal moves function $\textsc{Moves}(\delta_i, s_i)$ returns the set of configurations that may be transitioned to from (δ_i, s_i) by performing the first step of δ_i in the situation s_i. Search proceeds by building a tree (with root node (δ, s_0)), whose paths represent partial or complete executions of δ. For each fringe (set of leaf configurations) of this tree, *Fringe*, an evaluation function $\textsc{Eval}(\textit{Fringe}, \mathcal{P})$ selects which configuration in *Fringe* to transition to next given a dominance graph $\mathcal{P}$ over *Fringe*. If a final configuration (δ_r, s_r) is selected by EVAL, the execution represented by this configuration s_r forms the result of search.

Our aim is to allow an agent $a = \langle \mathbf{C}, \mathcal{I}\rangle$ to discover a final configuration representing an execution in $MPS(\mathcal{A})$ where $\mathcal{A}$ are the executions of δ given s_0. Algorithm 1 outlines the program interpretation process, where: EXPAND(*Fringe*, γ) removes the configuration $\gamma = (\delta_i, s_i)$ from the search fringe and replaces it with $\textsc{Moves}(\delta_i, s_i)$; ARGUE-ADF(*Fringe*, **C**, $\mathcal{I}$) determines the undefeated conclusions $\mathcal{P}$ of an *ADF* defined over the configurations in *Fringe*; and EVAL(*Fringe*, $\mathcal{P}$) selects a configuration in *Fringe* to transition to next based on the dominance graph $\mathcal{P}$.

Algorithm 1 PROGRAM INTERPRETATION PROCESS

```
Input: (δ, s0), C, I,    Output: Best
Best ← (δ, s0), Fringe ← {(δ, s0)}
while ¬Final(Best) do
    Fringe ← EXPAND(Fringe, Best)
    P ← ARGUE-ADF(Fringe, C, I)
    Best ← EVAL(Fringe, P)
end while
```

ARGUE-ADF(*Fringe*, C, $\mathcal{I}$) Each fringe in the search tree constructed by EXPAND consists of a set of configurations (δ_i, s_i) where each (δ_i, s_i) is reachable from (δ, s_0) by performing a sequence of actions in δ. ARGUE-ADF constructs a dominance graph over the configurations in a fringe, with the objective of providing advice to EVAL in its selection of the next configuration to transition to (expand).

An agent $a = \langle \mathbf{C}, \mathcal{I}\rangle$ determines $\mathcal{P}$ by forming an ADF (Definition 2) over the configurations in *Fringe*. Each argument in the ADF expresses a preference for one configuration $\gamma_1 \in \textit{Fringe}$ over another $\gamma_2 \in \textit{Fringe}$ (based on the executions or partial executions they represent), and describes a reason why γ_1 is more likely to lie on a path to a most preferred final configuration (execution) than γ_2.

If arguments in the ADF over *Fringe* are based solely on actions that have been performed on a path from (δ, s_0) to a configuration in *Fringe*, the search for a most preferred execution of δ becomes greedy. To achieve admissibility, we must ensure that an agent can construct an argument preferring one configuration $\gamma_1 \in Fringe$ to another $\gamma_2 \in Fringe$ if, according to **C**, γ_1 leads to a configuration that is preferred to γ_2. We do not want an agent to search ahead in a program to determine which actions are actually possible, as this would defeat the purpose of best first search. Instead, we allow an agent to *speculate* on (estimate) which actions *could* be performed after making a transition, and construct arguments accordingly.

To formally define a speculation, we introduce a relaxation of the transition semantics of [de Giacomo *et al.*, 2000]. In this relaxed semantics, a *TransR* axiom is defined for each of the constructs in Section 2. This semantics is defined such that: if a configuration γ can transition to a configuration γ' by *Trans*, γ can transition to γ' by *TransR*; and if a configuration γ cannot transition to a configuration γ' by *Trans*, it *may* transition to γ' by *TransR*. The relaxed semantics *overestimates* what is possible in a program.

For each construct defined in Section 2, with the exception of primitive actions, *TransR* and *Trans* are equivalent. Given a primitive action $\delta = a$, (δ, s) transitions as follows,

$$TransR(a, s, \delta', s') \equiv \delta' = nil \ \wedge \ s' = do(a[s], s)$$

and differs from its *Trans* counterpart in that it does not check if an action is possible.

Definition 4 (Speculation) *For a configuration γ = (δ', s'), $\gamma_s = (\delta_s, s_s)$ is a speculation of γ if it is final ($Final(\delta_s, s_s)$) and a member of the reflexive, transitive closure of TransR (TransR*) where $TransR^*$(δ', s', δ_s, s_s) holds if (δ_s, s_s) is reachable from (δ', s') by repeated application of TransR*[3].

With the introduction of speculations the set of arguments AR in the ADF defined over a fringe is formed as follows: for each pair of configurations on a fringe (γ_1, γ_2) (where both γ_1 and γ_2 are final or non-final, or γ_1 is non-final and γ_2 final), an argument $\langle c, \mathcal{P}(\gamma_1, \gamma_2)\rangle$ (as described in Section 4) is constructed in support of $\mathcal{P}(\gamma_1, \gamma_2)$ if a speculation of γ_1 is preferred to γ_2 according to a comparison principle $c \in$ **C**.

EVAL(*Fringe*, $\mathcal{P}$) EVAL computes a set $\mathcal{E}_{MPS}$ representing the configurations in *Fringe* an agent estimates to lie on a path from (δ, s_0) to a final configuration (δ', s') where s' $\in MPS(\mathcal{A})$ and $\mathcal{A}$ are the executions of δ. $\mathcal{E}_{MPS}$ is the result of applying a choice rule Ch to the dominance graph $\mathcal{P}$ induced by an ADF over *Fringe* in ARGUE-ADF:

$$\mathcal{E}_{MPS} \stackrel{\text{def}}{=} Ch(Fringe, \mathcal{P}) \subseteq Fringe$$

EVAL selects a configuration in *Fringe* to transition to next, using the advice in $\mathcal{E}_{MPS}$. As $\mathcal{P}$ is based on speculations that may not represent legally reachable configurations, some of the preferences in $\mathcal{P}$ are transient. A transient preference between configurations γ_1 and γ_2 does not exist between all configurations reachable from γ_1 and all those reachable from γ_2, limiting an agent's ability to predict the nature of a $\mathcal{P}$ over all final program configurations from $\mathcal{P}$ over a search fringe.

To address problems arising from transient preferences, EVAL is implemented using prioritised configuration selection. This prioritisation is based on the following assumption regarding the nature of all choice rules an agent may adopt.

Assumption 1 *Given a non-empty set of alternatives $\mathcal{D}$, and a dominance graph over those alternatives $\mathcal{P}$, any choice rule Ch adopted by an agent: (i) finds a non-empty subset of $\mathcal{D}$ where $Ch(\mathcal{D}, \mathcal{P})$ contains (at least) all alternatives $\gamma \in \mathcal{D}$ for which there is no alternative $\gamma' \in \mathcal{D}$ such that $\mathcal{P}(\gamma', \gamma)$; and (ii) satisfies the property that an alternative $\gamma \in \mathcal{D}$ being in $Ch(\mathcal{D}, \mathcal{P})$ is independent of any alternative $\gamma' \in \mathcal{D}$ not connected to γ by an undirected path in $\mathcal{P}$.*

Assumption 1 provides an agent with a stopping condition – an indication of when a final configuration in the $\mathcal{E}_{MPS}$ of a fringe represents a most preferred execution.

The Prioritisation Algorithm Given a search fringe *Fringe*, a dominance graph $\mathcal{P}$ over *Fringe*, and the set $\mathcal{E}_{MPS} = Ch(Fringe, \mathcal{P})$, EVAL(*Fringe*, $\mathcal{P}$) is defined to:

(i) Select a final configuration $\gamma \in \mathcal{E}_{MPS}$ for which $\neg\exists\gamma'$ $\in Fringe$. $\mathcal{P}(\gamma', \gamma)$, if one exists;

(ii) If *(i)* is not satisfied, select a final configuration $\gamma \in \mathcal{E}_{MPS}$ which is not connected by an undirected path of preferences in $\mathcal{P}$ to a non-final configuration $\gamma' \in$ *Fringe*, if one exists;

(iii) If *(i)-(ii)* are not satisfied, select a non-final configuration $\gamma \in \mathcal{E}_{MPS}$, if one exists; and

(iv) If *(i)-(iii)* are not satisfied, select any non-final configuration $\gamma' \in Fringe$.

Steps (i) and (ii) specify the conditions that must be satisfied by a final configuration to demonstrate that it represents an execution in $MPS(\mathcal{A})$. Transient preferences can only exist between configuration pairs in which at least one member is non-final. If a final configuration $\gamma \in Fringe$ is not independent of all members of such pairs (according to Assumption 1), an agent cannot predict the status of the execution it represents when a $\mathcal{P}$ is constructed over all executions in $\mathcal{A}$. In this case, steps (iii)-(iv) tell the agent to continue its search.

Theorem 1 *Given an agent $a = \langle \boldsymbol{C}, \mathcal{I}\rangle$, using a choice rule satisfying Assumption 1, any execution $s_r \in \mathcal{A}$ of a program δ, given an initial situation s_0, found as a result of argumentation-based best first search is in $MPS(\mathcal{A})$.*

Proof: We demonstrate that a final configuration $\gamma = (\delta_r, s_r)$ of δ cannot be selected from any search fringe *Fringe*, given a dominance graph over its members $\mathcal{P}$, if $s_r \notin MPS(\mathcal{A})$. EVAL, using the prioritisation algorithm, will only select $\gamma \in Ch(Fringe, \mathcal{P})$ as a result of search if: (i) there is no configuration $\gamma' \in Fringe$ such that $\mathcal{P}$(γ', γ); or (ii) there is no non-final configuration $\gamma' \in Fringe$ where γ and γ' are connected by an undirected path in $\mathcal{P}$.

If (i) holds, then there is no undefeated argument (based on a speculation or not) that can be formed in support of $\mathcal{P}(\gamma', \gamma)$

[3]We consider only Golog programs that admit a finite number of executions, and for which there is no situation in which iteration and loop constructs admit an infinite number of iterations.

for $\gamma' \in \mathit{Fringe}$. As speculation over-estimates what is possible after expanding a configuration, there is no final configuration of δ given s_0 that is preferred to γ. According to part (i) of Assumption 1, s_r must be a member of $MPS(\mathcal{A})$.

Let U be the set of configurations in *Fringe* connected to γ by an undirected path in $\mathcal{P}$. If (ii) holds, each configuration in U is final, and no configuration in U is connected to a non-final configuration by an undirected path in $\mathcal{P}$. Thus, there is no argument (based on a speculation) in support of a preference between a configuration $\gamma' \in \mathit{Fringe}$ and a configuration in U. As speculation over-estimates what is possible, there are no final configurations of δ outside of U that could be connected to γ (or any configuration in U) by an undirected path in $\mathcal{P}$. By part (ii) of Assumption 1, the question of whether s_r is in $MPS(\mathcal{A})$ is independent of any execution in $\mathcal{A}$ represented by a configuration outside of U. Let U' be the set of executions represented by configurations in U. As $U' \subseteq \mathcal{A}$, and s_r is in $MPS(U')$, s_r must be in $MPS(\mathcal{A})$.

Moreover, the assumption that $Ch(\mathit{Fringe}, \mathcal{P})$ is non-empty for a non-empty Fringe (Assumption 1) ensures that if a program δ has at least one legal execution, argumentation-based best first search is guaranteed to find a result. □

Example 3 (The Travelling Agent cont.) *In the first step of the interpretation of δ in Example 1:*

$$\mathit{Fringe} = \{\gamma_1 = (\delta', s_1'), \gamma_2 = (\delta', s_2')\}$$

where s_1' = do(drive(h, u), s_0), s_2' = do(walk(h, u), s_0), and δ' = [eat(cake) ; train(u, c)] | [skip(u, c) ; eat(fruit)]. The speculations that can be formed of (δ', s_1') and (δ', s_2') are $\{sp_1, sp_2\}$ and $\{sp_3, sp_4\}$, respectively:

$$sp_1 = (nil, do(train(u,c), do(eat(cake), s_1')))$$
$$sp_2 = (nil, do(eat(fruit), do(skip(u,c), s_1')))$$
$$sp_3 = (nil, do(train(u,c), do(eat(cake), s_2')))$$
$$sp_4 = (nil, do(eat(fruit), do(skip(u,c), s_2')))$$

*In an ADF defined over Fringe (given **C** and $\mathcal{I}$ as described in Examples 1 and 2), $AR = \{A_i\}_{i=1,2}$, where $A_1 = \langle \mathcal{C}_3, \mathcal{P}(\gamma_1, \gamma_2)\rangle$ and $A_2 = \langle \mathcal{C}_2, \mathcal{P}(\gamma_2, \gamma_1)\rangle$. A_1 is based on principle $\mathcal{C}_3$ (driving is less tiring than walking). A_2 is based on sp_4 and principle $\mathcal{C}_2$ (walking and skipping together is preferred over driving). A_2 defeats A_1 (by Definition 2 and $\neg\mathcal{I}(\mathcal{C}_3, \mathcal{C}_2)$), and $\mathcal{P} = \{(\gamma_2, \gamma_1)\}$.*

Given the Schwartz set [Schwartz, 1972] as Ch, EVAL *selects $walk$ as the first step ($\mathcal{E}_{MPS} = Ch(\mathit{Fringe}, \mathcal{P}) = \{\gamma_2\}$ and $\neg\exists\gamma \in \mathit{Fringe}.\mathcal{P}(\gamma, \gamma_2)$), resulting in the fringe:*

$$\mathit{Fringe} = \{\gamma_1 = (\delta', s_1'),$$
$$\gamma_3 = (train(u,c), do(eat(cake), s_2')),$$
$$\gamma_4 = (eat(fruit), do(skip(u,c), s_2'))\}$$

Performing the ARGUE-ADF *step on the new fringe results in a new set of arguments $AR = \{A_i\}_{i=1,\ldots,5}$ where: $A_1 = \langle \mathcal{C}_1, \mathcal{P}(\gamma_4, \gamma_3)\rangle$ (based on sp_4), $A_2 = \langle \mathcal{C}_2, \mathcal{P}(\gamma_4, \gamma_1)\rangle$ (based on sp_4), $A_3 = \langle \mathcal{C}_3, \mathcal{P}(\gamma_1, \gamma_3)\rangle$ (based on sp_1 and sp_2), $A_4 = \langle \mathcal{C}_3, \mathcal{P}(\gamma_1, \gamma_4)\rangle$ (based on sp_1 and sp_2), and $A_5 = \langle \mathcal{C}_1, \mathcal{P}(\gamma_1, \gamma_3)\rangle$ (based on sp_2). The conclusions of undefeated arguments in AR form the dominance graph $\mathcal{P} = \{(\gamma_4, \gamma_3), (\gamma_4, \gamma_1), (\gamma_1, \gamma_3)\}$.* EVAL *selects the configuration γ_4 to transition to, replacing γ_4 on the fringe with sp_4.*

Repeating the ARGUE-ADF *and* EVAL *steps given this new fringe results in the execution represented by sp_4 being chosen as the final result. In Example 2, $MPS(\mathcal{A} = \{s_1, s_2, s_3\}) = \{s_3\}$. In this example, $sp_4 = (nil, s_3)$.*

6 Conclusions

This paper has presented an argumentation-based best first search algorithm for the interpretation of Golog programs. Argumentation allows the flexible specification (and elicitation) of preferences (including comparative preference), while the incorporation of best first search reduces the execution space traversed in finding a most preferred result. Our approach uses the concept of speculation (a tool for predicting future opportunities at each stage of search), and arguments in support of preference between configurations, to guide search toward a most preferred program execution. The proposed argumentation-based interpretation of Golog programs is shown to be admissible. As future work, we intend to consider the implicit speculation of programs, in place of the explicit generation of speculations by a $TransR$ relation.

References

[Bar-Hillel and Margalit, 1988] M. Bar-Hillel and A. Margalit. *Theory and Decision*, 24:119–145, 1988.

[Bench-Capon, 2003] T. J. M. Bench-Capon. Persuasion in practical argument using value based argumentation frameworks. *J. of Log. Comp.*, 13:429–448, 2003.

[Bienvenu *et al.*, 2006] M. Bienvenu, C. Fritz, and S. McIlraith. Planning with Qualitative Temporal Preferences. In *KR*, pages 134–144, 2006.

[Boutilier *et al.*, 2000] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-Theoretic, High-Level Agent Programming in the Situation Calculus. In *AAAI*, 2000.

[de Giacomo *et al.*, 2000] G. de Giacomo, Y. Lespérance, and H. J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1-2):109–169, 2000.

[Delgrande *et al.*, 2007] James Delgrande, Torsten Schaub, and Hans Tompits. *J. of Log. Comp.*, 17(5):871–907, 2007.

[Fritz and McIlraith, 2006] Christian Fritz and Sheila McIlraith. Decision-Theoretic Golog with Qualitative Preferences. In *KR*, pages 153–163, 2006.

[Levesque *et al.*, 1997] Hector Levesque, Ray Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. Golog: A logic programming language for dynamic domains. *J. of Log. Prog.*, 31(1-3):59–83, 1997.

[Ouerdane *et al.*, 2007] W. Ouerdane, N. Maudet, and A. Tsoukias. Arguing over Actions That Involve Multiple Criteria: A Critical Review. In *ECSQARU*, 2007.

[Schwartz, 1972] T. Schwartz. Rationality and the myth of the maximum. *Noûs*, 6(2):97–117, 1972.

[Sohrabi *et al.*, 2006] S. Sohrabi, N. Prokoshyna, and S. McIlraith. In *ISWC*, pages 597–611, 2006.

[Son and Pontelli, 2004] T. Son and E. Pontelli. Planning with preferences using logic programming. In *LPNMR*, pages 247–260, 2004.

[Tversky, 1969] A. Tversky. Intransitivity of Preferences. *Psychological Review*, 76(1):31–48, 1969.