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 $A$ 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 $A$ ($MPS(A)$). We apply a choice rule, such as the Schwartz [Schwartz, 1972] criterion, to a dominance graph over $A$ 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 $\delta$, an instantiation of our $ADF$ provides an estimate of which available transitions will lead to an execution in $MPS(A)$, where $A$ denotes the executions of $\delta$. 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 $A$ (provided $A \neq \emptyset$).

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(𝒜) for a set of program executions 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); φ? (test that φ holds in the current situation); δ₁; δ₂ (execute program δ₁ then δ₂); δ₁|δ₂ (choose to execute program δ₁ or δ₂); πνφ (choice of arguments v in δ); δ∗ (execute δ zero or more times); if φ then δ₁ else δ₂ (synchronised conditional); while φ do δ (synchronised loop); and proc P(π)δ 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 δ such that: D |= do(δ, s₀, do(δ, s₀)), where do(δ, s₀) is a legal terminating situation (execution) of δ when executed in the situation s₀. 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, δ₁; δ₂, a Trans axiom is defined to determine which configurations can be transitioned to from (δ₁; δ₂, s) by performing a single step of δ₁ in situation s. A configuration (δ₁; s) is final (Final(δ₁; s)) if no steps remain to be performed in δ₁.

A selection of Trans and Final axioms are shown below². Given a program of the form δ₁ = a (where a is a primitive action), the configuration (δ₁; s) can transition if it is possible to perform action a in situation s (Poss(a[s], s)).

\[
\text{Trans}(a, s, δ', s') \equiv \\
\text{Poss}(a[s], s) \land δ' = \text{nil} \land s' = \text{do}(a[s], s)
\]

and is not final as the action a remains to be performed. For the sequence δ₁ = δ₁; δ₂ (where δ₁ is performed first and then δ₂), the configuration (δ₁; s) transitions as follows.

\[
\text{Trans}(δ₁; δ₂, s, δ', s') \equiv \exists γ. δ' = (γ; δ₂) \land \\
\text{Trans}(δ₁, s, γ, s') \lor \text{Final}(δ₁, s) \land \text{Trans}(δ₂, s, δ', s')
\]

and is final if both δ₁ and δ₂ are final in s:

\[
\text{Final}(δ₁; δ₂, s) \equiv \text{Final}(δ₁, s) \land \text{Final}(δ₂, s)
\]

²do(\{a₁, a₂, ..., aₙ₋₁, aₙ\}, s) is shorthand for do(aₙ, do(aₙ₋₁, do(a₂, do(a₁, s)))).

³A 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 PP [Son and Pontelli, 2004]. 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 ∧, ∨, and ¬. 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 \( \delta \).

We first provide the following definition of a comparison principle, and assume that an agent \( a = (\mathcal{C}, \mathcal{I}) \) is equipped with a finite set of such principles \( \mathcal{C} \), and a (transitive, irreflexive, and asymmetric) partial ordering \( \mathcal{I} \) over the principles in \( \mathcal{C} \). Given \( c_1, c_2 \in \mathcal{C} \), \( \mathcal{I}(c_1, c_2) \) 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 \( \phi \) 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 \( \delta \)) 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.

```plaintext
proc travel(A, B, C)
    [drive(A, B)|walk(A, B)];
    [[eat(cake); train(B, C)] | [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 \( s_0, A = \text{home} (h), B = \text{university} (u), C = \text{cinema} (c) \), the program \( \delta \) has three executions:

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

Agent \( a \) has three principles \( C_1(s_1, s_2, \phi_1) \) by which it compares these executions. By the first comparison principle, agent \( a \) prefers one execution over another if it involves eating tastier food:

\[
\phi_1(\text{do}(a_1^1, s_0), \text{do}(a_2^2, s_0)) \equiv \text{eat}(x) \in a_1^1 \land \text{eat}(y) \in a_2^2 \land x >_{\text{taste}} y
\]

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

\[
\phi_2(\text{do}(a_1^1, s_0), \text{do}(a_2^2, s_0)) \equiv \text{walk}(x_i, y_i) \in a_1^1 \land \text{skip}(x_j, y_j) \in a_2^2
\]

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

\[
\phi_3(\text{do}(a_1^1, s_0), \text{do}(a_2^2, s_0)) \equiv x \in a_1^1 \land y \in a_2^2 \land x <_{\text{tiring}} y
\]

where \( x <_{\text{tiring}} y \) 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 $s_i \in A$ is preferred to another $s_j \in A (P(s_i, s_j))$. Each argument takes the form $⟨J, P(s_i, s_j)⟩$ where $s_i, s_j \in A$ and $J$ is a justification for the truth of the conclusion $P(s_i, s_j)$. Each justification $J$ represents the instantiation of a comparison principle $c \in C$ used by an agent to determine if one execution is preferred to another. An argument $a_1 = \langle J_1, P(s_i, s_j) \rangle \in AR$ is attacked (rebutted) by all arguments with an opposing conclusion $a_2 = \langle J_2, P(s_i, s_j) \rangle \in AR$ (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) $P$ over the available decisions (executions).

**Definition 2** An ADF is a 5-tuple: $ADF = (AR, attacks, C, pri, T)$ 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 $T \subseteq C \times C$ 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 $¬I(pri(b), pri(a))$ holds.

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

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

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 $\rightarrow$ taste, cake $\rightarrow$ tiring walk, and $T = \{⟨C_1, C_2⟩, ⟨C_1, C_3⟩, ⟨C_2, C_3⟩\}$. In an ADF defined over $A = \{s_1, s_2, s_3\}$, $a$ is able to instantiate its comparison principles to produce five arguments in support of preference relationships between elements of $A (AR = \{A_i\}_{i=1, \ldots, 5})$.

Argument 1, $A_1 = \langle C_1, P(s_1, s_2) \rangle$, denotes that $s_1$ is preferred to $s_2$ as it involves eating tastier food. Similarly, $A_2 = \langle C_1, P(s_3, s_2) \rangle$. Argument 3, $A_3 = \langle C_2, P(s_3, s_1) \rangle$, denotes that $s_3$ is preferred to $s_1$; and $s_3$ involves walking and skipping while $s_1$ involves driving. Argument 4, $A_4 = \langle C_3, P(s_1, s_2) \rangle$, denotes that $s_1$ is preferred to $s_2$ as it is less tiring. Similarly, $A_5 = \langle C_3, P(s_3, s_1) \rangle$.

By Definition 2, $A_3$ and $A_2$ attack each other with $A_3$ defeating $A_2$ (as $¬I(C_3, C_2)$), and $P = \{⟨s_1, s_2⟩, ⟨s_3, s_2⟩, ⟨s_3, s_1⟩\}$. Given the Schwartz set (Definition 3) as $Ch_i$, the most preferred subset of $A$, $MPS(A)$, is $Ch(A, P) = \{s_3\}$.

## 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 $(δ_i, si)$ that can be transitioned to from $(δ_0, s_0)$ by performing a sequence of actions in $δ$. Given a configuration $(δ_i, s_i)$, a legal moves function $MOVES(δ_i, s_i)$ 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 $(δ_0, s_0)$), whose paths represent partial or complete executions of $δ$. For each fringe (set of leaf configurations) of this tree, Fringe, an evaluation function $EVAL(Fringe, δ)$ selects which configuration in Fringe to transition to next given a dominance graph $P$ over Fringe. If a final configuration $(δ_f, s_f)$ is selected by EVAL, the execution represented by this configuration $s_f$ forms the result of search.

Our aim is to allow an agent $a = (C, T)$ to discover a final configuration representing an execution in $MPS(A)$ where $A$ are the executions of $δ$ given $s_0$. Algorithm 1 outlines the program interpretation process, where: $EXPAND(Fringe, γ)$ removes the configuration $γ = (δ_i, s_i)$ from the search fringe and replaces it with $MOVES(δ_i, s_i)$; $ARGUE-ADF(Fringe, C, T)$ determines the undeclared conclusions $P$ of an ADF defined over the configurations in Fringe; and $EVAL(Fringe, P)$ selects a configuration in Fringe to transition to next based on the dominance graph $P$.

### Algorithm 1 Program Interpretation Process

**Input:** $(δ_0, s_0), C, T$. **Output:** Best

`Best ← (δ_0, s_0), Fringe ← \{δ_0\}`

while ¬Final(Best) do

`Fringe ← EXPAND(Fringe, Best)`

`P ← ARGUE-ADF(Fringe, C, T)`

`Best ← EVAL(Fringe, P)`

end while

**ARGUE-ADF(Fringe, C, T)** 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 $(δ_0, s_0)$ 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 $a = (C, T)$ determines $P$ by forming an ADF (Definition 2) over the configurations in Fringe. Each argument in the ADF expresses a preference for one configuration $γ_1 \in Fringe$ over another $γ_2 \in Fringe$ (based on the ex-
executions or partial executions they represent), and describes a reason why $\gamma_1$ is more likely to lie on a path to a most preferred final configuration (execution) than $\gamma_2$.

If arguments in the ADF over Fringe are based solely on actions that have been performed on a path from $(\delta, s_0)$ to a configuration in Fringe, the search for a most preferred execution of $\delta$ 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 $C$, $\gamma_1$ leads to a configuration that is preferred to $\gamma_2$. 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 $\gamma$ can transition to a configuration $\gamma'$ by Trans, $\gamma$ can transition to $\gamma'$ by TransR; and if a configuration $\gamma$ cannot transition to a configuration $\gamma'$ by $\gamma$, it may transition to $\gamma'$ 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 $\delta = (a, (\delta, s))$ transitions as follows:

$$TransR(a, s, s', s') \equiv s' = \text{nil} \land s' = do(a[s], s)$$

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

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

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

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

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

Eval selects a configuration in Fringe to transition to next, using the advice in $\mathcal{E}_{\mathcal{MPS}}$. As $\mathcal{P}$ is based on specifications that may not represent legally reachable configurations, some of the preferences in $\mathcal{P}$ are transient. A transient preference between configurations $\gamma_1$ and $\gamma_2$ does not exist between all configurations reachable from $\gamma_1$ and all those reachable from $\gamma_2$, 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 $P(\gamma', \gamma)$; 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 $\gamma$ 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 $\mathcal{E}_{\mathcal{MPS}}$ 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 $\mathcal{E}_{\mathcal{MPS}} = Ch(\text{Fringe}, \mathcal{P})$, Eval(Fringe, $\mathcal{P}$) is defined to:

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

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

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

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

Steps (i) and (ii) specify the conditions that must be satisfied by a final configuration to demonstrate that it represents an execution in $\mathcal{MPS}(\mathcal{A})$. Transient preferences can only exist between configuration pairs in which at least one member is non-final. If a final configuration $\gamma \in \text{Fringe}$ 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 $\alpha = (C, \mathcal{T})$, using a choice rule satisfying Assumption 1, any execution $s_r \in \mathcal{A}$ of a program $\delta$, given an initial situation $s_0$, found as a result of argumentation-based best first search is in $\mathcal{MPS}(\mathcal{A})$.

**Proof:** We demonstrate that a final configuration $\gamma = (\delta_r, s_r)$ of $\delta$ cannot be selected from any search fringe Fringe, given a dominance graph over its members $\mathcal{P}$, if $s_r \notin \mathcal{MPS}(\mathcal{A})$. 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 $\gamma' \in \text{Fringe}$ such that $P(\gamma', \gamma)$; or (ii) there is no non-final configuration $\gamma' \in \text{Fringe}$ where $\gamma$ and $\gamma'$ 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 $P(\gamma', \gamma)$.
for $\gamma' \in Fringe$. As speculation over-estimates what is possible after expanding a configuration, there is no final configuration of $\delta$ given $s_0$ that is preferred to $\gamma$. According to part (i) of Assumption 1, $s_r$ must be a member of $MPS(A)$.

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

Moreover, the assumption that $Ch(Fringe, P)$ is non-empty for a non-empty $Fringe$ (Assumption 1) ensures that if a program $\delta$ 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 $\delta$ in Example 1:

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

where $s_1' = do(drive(h, u))$, $s_0$), $s_2' = do(walk(h, u), s_0)$, and $\delta' = (eat(cake) ; train(u, c)) | (skip(u, c) ; eat(fruit))$. The speculations that can be formed of $\delta'$ and $s_2'$ 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 $I$ as described in Examples 1 and 2), $AR = \{A_1\}_{i=1,2}$, where $A_1 = (C_1, P(\gamma_1, \gamma_2))$ and $A_2 = (C_2, P(\gamma_1, \gamma_2))$. $A_1$ is based on principle $C_1$ (driving is less tiring than walking). $A_2$ is based on $sp_4$ and principle $C_2$ (walking and skipping together is preferred over driving). $A_2$ defeats $A_1$ (by Definition 2 and $\neg I(C_2, C_1)$, and $P = \{(\gamma_1, \gamma_2)\}$.

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

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

**Performing the ARGUE-ADF step on the new fringe results in a new set of arguments** $AR = \{A_1\}_{i=1,2,3,4,5}$ where:

- $A_1 = (C_1, P(\gamma_1, \gamma_3))$ (based on $sp_4$)
- $A_2 = (C_2, P(\gamma_1, \gamma_4))$ (based on $sp_4$)
- $A_3 = (C_3, P(\gamma_1, \gamma_3))$ (based on $sp_4$ and $sp_2$)
- $A_4 = (C_4, P(\gamma_1, \gamma_4))$ (based on $sp_4$ and $sp_2$)
- $A_5 = (C_5, P(\gamma_1, \gamma_3))$ (based on $sp_2$)

The conclusions of unpreferred arguments in $AR$ for the dominance graph $P = \{(\gamma_4, \gamma_3), (\gamma_4, \gamma_1), (\gamma_1, \gamma_3)\}$ $Eval$ selects the configuration $\gamma_4$ to transition to, replacing $\gamma_4$ 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, $MPS(A = \{s_1, s_2, s_3\}) = \{s_3\}$. In this example, $sp_4 = (\text{nil}, s_3)$.

### 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