Optimising the Interpretation of Golog Programs with Argumentation

Michelle Blom

NICTA Victoria Research Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne
Victoria 3010, Australia

Abstract. This paper presents some preliminary work toward the development of an argumentative interpreter for Golog programs [7]. Traditional Golog interpreters are not designed to find the most preferred executions of a program from the perspective of an agent. Existing techniques that optimise the interpretation of Golog programs suffer from either a lack of flexibility in how preference may be expressed or require the consideration of all program executions. The presented work combines the use of argumentation to compare executions relative to a set of general comparison criteria, and the theory behind informed search techniques to reduce the cost of the search process. The use of this argumentative informed search in the interpretation of ConGolog programs (with true concurrency) [3], and its potential to find a program execution that is the most preferred by a collection of agents, is also discussed.

1 Introduction

Agent programming languages such as Golog [7] are designed to define and determine a legal execution (or set of executions) of a program in the presence of non-determinism. Each legal execution represents one possible way of achieving a particular goal. Golog interpreters are designed to find *possible* executions of a program, not to identify the *best* executions – those that are the most preferred, or associated with the greatest utility. Several approaches have been designed to complement agent programming with both quantitative and qualitative preferences.

In [2], Boutilier *et al* develop DTGolog – a decision theoretic extension of Golog. DTGolog combines the theory of Markov decision processes (MDPs) with the Golog programming language. A DTGolog interpreter finds a policy for the execution of a program with the highest expected utility. This policy outlines, for each situation that may be transitioned to in the program, the best action to perform. In [5], Fritz and McIlraith integrate qualitative and quantitative preferences to form an interpreter for Golog programs. Qualitative preferences are compiled into a Golog program which, when executed synchronously with an agent program δ, returns the executions of δ that satisfy the qualitative preferences. The most quantitatively preferred executions amongst the qualitatively preferred are selected as the best interpretations of a program.

Existing techniques developed to improve the quality of legal executions found from agent programs have several limitations. First, many of these techniques require the traversal of all possible program executions, which can become intractable. Second, each technique defines a mechanism by which legal executions are compared, and encodes this into the transition or evaluation semantics of its interpreter. DTGolog, for example, requires the availability of numerical utilities and rewards associated with situations. [5] may only be used where qualitative constraints are hard and more important than quantitative preferences. Addressing these limitations requires the development of an interpreter for Golog programs that allows for a flexible specification of comparison criteria, and an approach for reducing the space of legal executions to be compared.

This paper presents preliminary work toward the development of an argumentative Golog interpreter. It combines the use of argumentation to compare executions relative to a set of general comparison criteria, and the theory behind informed search techniques to reduce the cost of the search process. To the best of our knowledge, this is the first attempt to combine argumentative techniques with Golog programming.

Considered first is the use of a tournament-based argumentation framework to construct preference relationships over choices. We combine this framework with voting theory to select the most preferred executions from an execution set $\mathcal{A}$. We assume that there are *reasons* why one execution is preferred to another, where these reasons are based on aspects of an execution, and not necessarily the execution as a whole. Reasons may be simultaneously constructed in support of an execution $\gamma_1 \in \mathcal{A}$ being preferred over an execution $\gamma_2 \in \mathcal{A}$ ($\mathcal{P}(\gamma_1, \gamma_2)$), and in support of the execution γ_2 being preferred over execution γ_1 ($\mathcal{P}(\gamma_2, \gamma_1)$). Analysis and debate over these reasons allows an agent to determine what preference relationships exist between pairs of executions in the set $\mathcal{A}$.

The conclusions of undefeated arguments represent a preference graph, in which the vertices are the executions of $\mathcal{A}$, and the directed arcs are preference relationships between executions. We describe how a voting system criterion can be used to take this set of preference relationships and determine an optimal subset of $\mathcal{A}$, demonstrating the approach with the Schwartz set criterion [9]. We additionally discuss the benefit of combining argumentation and voting in separate stages, as opposed to developing an argumentation framework that computes optimal outcomes as its acceptable conclusions.

This argumentative approach for comparing executions does not help an agent reduce the search cost involved in finding the best execution of a program. It still needs to generate all executions in order to compare them. We consider the use of informed search and argumentation over incomplete executions to argumentatively select which choices to make in the interpretation of a program. Executions represent paths in a search tree, and an argumentative evaluation function is used to select which choices (nodes in the tree) will lead to paths representing the most preferred solutions.

As choices are made (and nodes are expanded), a preference graph amongst unexpanded nodes is constructed, its optimal set is determined (for example,

with the Schwartz set criterion [9]), and nodes are selected from this set for the next expansion. On its own, this algorithm is an example of greedy search – it does not consider what may be possible after expanding a node – and will not necessarily lead to an optimal result.

Given a program δ, our aim is to find an execution that is in the optimal subset of $\mathcal{A}$ (the complete set of executions of δ). To achieve this, we allow an agent to speculate on what actions *could* be performed after expanding a node, without requiring a search ahead to determine what is *actually* possible. An agent is then able to use these speculations as grounds for preferring one unexpanded node over another. The optimal subset of a set of unexpanded nodes is then computed based on both speculative and non-speculative arguments. Conditions upon the voting criteria employed in the approach, and prioritised node selection from an optimal set of unexpanded nodes, are defined to ensure that transient arguments – based on false speculations – do not cause sub-optimal executions to be found as a result.

The remainder of this paper is structured as follows. In Section 2, the Golog programming language and the situation calculus are described. Section 3 discusses how argumentation may be used to select the most preferred executions from an execution set based on a set of general comparison criteria. In Section 4, we consider how the interpretation of a Golog program may be performed using an argumentative informed search technique. In Section 5, a sketch of a proof demonstrating soundness of the approach is given. How multiple agents may use argumentation to collaboratively find an execution of a ConGolog program is discussed in Section 6.

2 Golog, ConGolog, and The Situation Calculus

Golog programs are described using axioms of the situation calculus [3], together with operators and constructs to encode procedures, actions, iteration, and non-deterministic choice. The situation calculus is a formalism for reasoning about dynamically changing domains or worlds, developed by McCarthy in [8]. The situation calculus enables reasoning about: *actions*, which change the state of the world; *situations*, which describe a sequence of actions applied to an initial state; and *fluents*, which represent properties of the world.

The basic constructs present in the Golog programming language are described in [3, 6]. Given a domain theory D, consisting of a situation calculus theory of action, a legal execution of a program δ is a sequence of actions $\boldsymbol{a}$ such that: $D \models Do(\delta, s_o, do(\boldsymbol{a}, s_o))$, where $Do(\delta, s, s')$ indicates that s' is a legal terminating situation when the program δ is executed starting in situation s.

Each program-situation pair is called a configuration. In [3], a transition semantics is defined which determines whether an agent can transition from one configuration (δ, s) to another (δ', s') by performing one step in a program δ, resulting in a situation s' and program left to execute δ'. For each of the Golog constructs defined in [3, 6], *Trans* and *Final* predicates are defined to indicate if a transition between two configurations is possible by executing a program statement of that type, and whether a program together with a situation

represents a final configuration. A final configuration is one in which no further steps need to be performed and represents a legal execution of a program. A complete specification of the transition semantics may be found in [3].

A legal execution of a program δ in a situation s may be found by repeatedly applying the transition relation *Trans* starting at the configuration (δ, s). The set of all possible configurations (δ', s') that may be reached in this manner is denoted by $Trans^*(\delta, s, \delta', s')$. The *Do* relation is defined as:

$$Do(\delta, s, s') \stackrel{\text{def}}{=} \exists \delta'. Trans^*(\delta, s, \delta', s') \wedge Final(\delta', s')$$

ConGolog is an extension of Golog incorporating additional programming constructs to handle the concurrent execution of multiple programs [3]. True concurrency is achieved by allowing actions from concurrently executing programs to be performed at the same time. ConGolog introduces several new programming constructs, described in [3].

3 Argumentation over Configurations

To find the optimal subset of a set of choices $\mathcal{A}$ requires a definition of optimality – how does one decide if a choice $\gamma_1 \in \mathcal{A}$ is optimal or not? This classification is, necessarily, based on the preferences that exist between pairs of choices.

In rational choice theory, it is assumed that each decision maker has a fixed, stable, and complete set of preferences over its available choices. Utility maximisation over these preferences produces a complete and transitive ranking from which an optimal choice – a choice with the highest ranking – may be selected. Critics of rational choice theory advocate a less restrictive and more human-like method of decision-making based on reasons [10, 11], noting that a decision maker often does not have such a preference set. In these alternative theories, preference amongst choices 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.

Our aim is to provide an agent with the flexibility to compare executions based on whichever principles or criteria they desire. These criteria – of varying importance to a decision maker – may be qualitative in the form of values, quantitative in the form of cost minimisation, or decision rules accumulated from experience. As such, we follow the reason-based method of decision making and define an argumentation framework for the construction of preferences.

We define a tournament-based argumentation framework (*TAF*) for the construction of preferences amongst a set of choices $\mathcal{A}$. These choices represent the set of final configurations for a program δ given an initial situation s_0. The structure of a *TAF* is defined based on the structure of an audience-specific value-based framework (*AVAF*) [1]. In [1], Bench-Capon defines an *AVAF* as an extension of the Dung argumentation framework [4]. In a Dung argumentation framework, $\langle AR, attacks \rangle$, a finite set of arguments AR are constructed in support of and against a set of beliefs. Given an attack relation over those arguments, *attacks*, an extension of the framework is computed to represent the

set of acceptable arguments in AR and, consequently, the conclusions to be believed. In [1], some conflicts can be resolved by considering the values promoted by each argument and the relative importance of those values. An argument $a_1 \in \mathcal{A}$ in support of (or against) a choice, for example, is said to defeat an opposing argument $a_2 \in \mathcal{A}$ if $attacks(a_1, a_2)$ holds and the value promoted by a_2 is not more important than the value promoted by a_1.

Each argument in a TAF expresses a reason why one configuration $\gamma_1 \in \mathcal{A}$ is preferred to another $\gamma_2 \in \mathcal{A}$ ($\mathcal{P}(\gamma_1, \gamma_2)$). Each argument takes the form $\langle J, \mathcal{P}(\gamma_1, \gamma_2)\rangle$ where $\gamma_1, \gamma_2 \in \mathcal{A}$ and J is a justification or support for the truth of the conclusion $\mathcal{P}(\gamma_1, \gamma_2)$. Each justification J represents the instantiation of a comparison criteria used by an agent to determine if one configuration is preferred to another. An argument $a_1 = \langle J_1, \mathcal{P}(\gamma_1, \gamma_2)\rangle \in AR$ is attacked by all arguments with an opposing conclusion $a_2 = \langle J_2, \mathcal{P}(\gamma_2, \gamma_1)\rangle \in AR$. An argument a_1 defeats an argument a_2 iff $attacks(a_1, a_2)$ holds and the criteria that derives a_1 is more important to the decision maker than the criteria that derives a_2.

Definition 1 *A tournament-based argumentation framework (TAF) is a 5-tuple:*

$$TAF = \langle AR, attacks, \mathcal{C}, cri, \mathcal{I}\rangle$$

where AR *is a finite set of arguments,* attacks *is an irreflexive binary attack relation on* AR*, $\mathcal{C}$ is a finite set of comparison criteria,* cri *maps arguments in* AR *to the criteria in $\mathcal{C}$ they are derived from, and $\mathcal{I} \subseteq \mathcal{C} \times \mathcal{C}$ is a (transitive, irreflexive, and asymmetric) partial ordering of the criteria in $\mathcal{C}$. An argument* a $\in$ AR *defeats an argument* b $\in$ AR *iff* attacks(a, b) *and* $\mathcal{I}$(cri(a), cri(b)) *holds.*

One can imagine a TAF as consisting of a tournament between each pair of choices in a set $\mathcal{A}$. For each pair of choices, (γ_1, γ_2), they either decide not to fight (no arguments are constructed in support of $\mathcal{P}(\gamma_1, \gamma_2)$ or $\mathcal{P}(\gamma_2, \gamma_1)$), they fight and one choice is victorious (either $\mathcal{P}(\gamma_1, \gamma_2)$ or $\mathcal{P}(\gamma_2, \gamma_1)$ holds), or they fight and the result is a draw (both $\mathcal{P}(\gamma_1, \gamma_2)$ and $\mathcal{P}(\gamma_2, \gamma_1)$ hold). Given a set of choices $\mathcal{A}$, the conclusions of undefeated arguments in a TAF represent a preference relation $\mathcal{P} : \mathcal{A} \times \mathcal{A}$. $\mathcal{P}$ can be represented as a graph, in which the vertices are the choices in $\mathcal{A}$, and the directed arcs are preference relationships amongst these choices.

Our notion of defeat varies from that defined in an $AVAF$ [1], as opposing arguments based on equally preferred criteria do not defeat each other – we allow two opposing statements $\mathcal{P}(\gamma_1, \gamma_2)$ and $\mathcal{P}(\gamma_2, \gamma_1)$ to coexist as acceptable conclusions. For this reason, the set of undefeated arguments – the arguments we view as acceptable in a TAF – does not equate to an extension (preferred or grounded) according to the semantics of Dung [4]. Any decision made between the two choices γ_1 and γ_2 is delegated to the voting stage, where the voting criterion used is able distinguish between ties arising from conflicting arguments, and those arising from incomparability. This distinction becomes important in Section 4, where we develop an interpreter for Golog programs based on this framework.

We assume that an agent has its own criterion for selecting an optimal subset of $\mathcal{A}$ based on the preferences that have been found to exist between its elements. A choice function $Ch(\mathcal{A}, \mathcal{P}) \subseteq \mathcal{A}$ is defined which, when given a set of configurations $\mathcal{A}$ and a preference relation $\mathcal{P}$, returns the subset of $\mathcal{A}$ representing optimal choices – the choices that are the most preferred. Whichever criterion is used to determine $Ch(\mathcal{A}, \mathcal{P})$, it must be able to manage cycles in the graph appropriately.

One criterion we may use to represent $Ch(\mathcal{A}, \mathcal{P})$ is the Schwartz set [9], which is defined as follows:

Definition 2 *Given a set of choices $\mathcal{A}$, and a preference relation over those choices $\mathcal{P}$:*

(i) A choice $\gamma \in \mathcal{A}$ is strictly preferred to a choice $\beta \in \mathcal{A}$ iff $(\gamma, \beta) \in \mathcal{P}$ and $(\beta, \gamma) \notin \mathcal{P}$.

(ii) A Schwartz set component of $\mathcal{A}$ is a subset $S \subseteq \mathcal{A}$ such that:

(a) No choice $\gamma \in \mathcal{A}$ where $\gamma \notin S$ is strictly preferred to a choice $\beta \in S$.

(b) No non-empty proper subset of S satisfies property (a).

(iii) The Schwartz set Sc of $\mathcal{A}$ is the union of all Schwartz set components of $\mathcal{A}$.

The Schwartz set criterion recognises that if a preference cycle exists between a set of choices, then these choices are essentially "indistinguishable" – it is a "matter of indifference which is chosen" [9]. The Schwartz set Sc for a set of choices $\mathcal{A}$ is the largest subset of $\mathcal{A}$ for which (i) no choice outside Sc is strictly preferred to a choice inside, and (ii) every choice in Sc is at least as good as every other choice in Sc.

Separating the construction of preferences (by argumentation) and the selection of a result (by voting theory) into two stages has its advantages. An abstract argumentation framework is designed to make binary decisions – should conclusion x be believed or not?, or, is choice γ_1 preferred to choice γ_2 or vice versa? To achieve our purpose in a single stage, arguments must be constructed in support of a choice being optimal or sub-optimal, for each of the choices available. The conditions associated with the adopted criterion for optimality must then be shown to be satisfied by constructing arguments in support of, and defeating arguments against, their satisfaction. For example, assume a choice γ_1 is considered optimal if no alternative choice γ_2 is preferred to it. Arguments attacking the satisfaction of this criteria must demonstrate that there is such a choice γ_2 that is preferred to γ_1 according to a given comparison criteria. These attacks may be defended by demonstrating that, in fact, γ_1 is preferred to γ_2 according to a more important criteria.

A disadvantage of the aforementioned approach is that the modularity of the two-stage process is lost. The same ultimate behaviour is being achieved in both the one and two stage processes – as what is under dispute is the preferences between choices – but additional work is required to transform an algorithm that computes optimal sets based on preferences into an argumentative setting. The result is that a decision maker cannot easily switch between different algorithms

for finding an optimal set based on a given criteria, or between different criteria for determining optimal sets.

At this preliminary stage in our work, we do not consider factual disputes (regarding the applicability of comparison criteria to choice pairs) as possible. As in an *AVAF* [1], arguments based on fact – which arise due to uncertainty in an agent's knowledge base and inferences – can be introduced in an *TAF* without modification to the framework. Assuming such uncertainty has implications not only on argumentation over preferences, but on what actions are considered possible in any given situation. Consequently, this uncertainty will have an impact on the program interpretation process described in Section 4.

The Traveling Agent Example

Consider an agent a with the following set of comparison criteria $\mathcal{C} = \{c_1, c_2, c_3\}$ where: c_1 = "*I prefer configurations that promote exercise* (Ex) *to those that demote it*", c_2 = "*I prefer configurations that promote efficiency* (Ef) *to those that demote it*", c_3 = "*I prefer configurations that promote safety* (Sa) *to those that demote it*", and $\mathcal{I}$ expresses that $c_3 >$ (is preferred to) $c_1 > c_2$. Assume that agent a needs to travel from its home to its university, and has the following program outlining how to travel from point A to point B.

proc travel(A, B)
    drive(A, B) | walk(A, B) | jog(A, B)
endProc

Assume that in situation s_0 each of the actions *drive*, *walk*, and *jog* are possible between $A = home$ and $B = uni$. $\mathcal{A}$ consists of three final configurations of the program for $A = home$ and $B = uni$:

$$\gamma_1 = (nil, do(drive, s_0))$$
$$\gamma_2 = (nil, do(walk, s_0))$$
$$\gamma_3 = (nil, do(jog, s_0))$$

According to agent a: *walk* promotes exercise and safety but demotes efficiency; *drive* promotes efficiency but demotes exercise; and *jog* demotes safety. Agent a is able to construct the following arguments in support of preference relationships between configurations of $\mathcal{A}$, where: $in(x, \gamma)$ denotes that action x has been performed in configuration γ, $p(x, y)$ denotes that action x promotes y, and $d(x, y)$ denotes that action x demotes y.

$$a_1 = \langle \{in(\gamma_1, drive), p(drive, Ef), in(\gamma_2, walk), d(walk, Ef), c_2\}, \mathcal{P}(\gamma_1, \gamma_2)\rangle$$
$$a_2 = \langle \{in(\gamma_1, drive), d(drive, Ex), in(\gamma_2, walk), p(walk, Ex), c_1\}, \mathcal{P}(\gamma_2, \gamma_1)\rangle$$
$$a_3 = \langle \{in(\gamma_3, jog), d(jog, Sa), in(\gamma_2, walk), p(walk, Sa), c_3\}, \mathcal{P}(\gamma_2, \gamma_3)\rangle$$

Given $AR = \{a_1, a_2, a_3\}$, the *attacks* relation contains the entries $attacks(a_1, a_2)$ and $attacks(a_2, a_1)$ from which an agent can derive that a_2 *defeats* a_1 based on our definition of a *TAF* (Definition 1). In the traveling agent example, the

configuration γ_2 is strictly preferred to the configurations γ_1 and γ_3. The optimal set of configurations for agent a to choose from, according to the Schwartz set criterion, is therefore $Ch(\{\gamma_1, \gamma_2, \gamma_3\}, \mathcal{P}) = \{\gamma_2\}$.

4 Argumentation as Informed Search (AIS)

Search algorithms are classified as either uninformed or informed based on how they traverse the state-space of a problem, represented as a search tree. Informed search algorithms use an evaluation function to determine which nodes in the tree to expand next in the search for a solution. This evaluation function ranks the desirability of expanding each node, and selects the node which is the most desirable for expansion. A node is more desirable than another if it is more likely to lead to a solution or the best solution.

We represent the interpretation of a Golog program in terms of informed search as follows. Using the transition semantics defined in [3] applied to Golog programs, and given a program δ to interpret with an initial situation s_0, our initial state is the configuration (δ, s_0). The state-space of the problem is all the configurations (δ_i, s_i) that may be transitioned to from (δ, s_0) by performing a sequence of actions in δ. Given a configuration (δ_i, s_i), a legal move function $\textsc{Moves}(\delta_i, s_i)$ returns the set of configurations that may be transitioned to from (δ_i, s_i) by performing a single action from the program δ_i in the situation s_i. The goal test is satisfied by a configuration (δ_i, s_i) if it is final $Final(\delta_i, s_i)$. Given a set of unexpanded nodes (configurations), Q, the evaluation function $\textsc{Eval}(Q, \mathcal{P})$ selects which configuration in Q to expand next given a preference relation $\mathcal{P}$ over Q.

Algorithm 1 GreedyArgue

Input: (δ, s_0), $\mathcal{C}$, $\mathcal{I}$
Output: *Best*
Best ← (δ, s_0)
Fringe ← $\{(\delta, s_0)\}$
while ¬Final(*Best*) **do**
  Fringe ← Expand(*Fringe*, *Best*)
  $\mathcal{P}$ ← Argue-TAF(*Fringe*, $\mathcal{C}$, $\mathcal{I}$)
  Best ← Eval(*Fringe*, $\mathcal{P}$)
end while

Fig. 1: The GreedyArgue algorithm.

Our aim is to allow an agent to find a final configuration of a program δ, given an initial situation s_0, that is in the optimal subset (according to an adopted criterion such as the Schwartz set [9]) of all final configurations $\mathcal{A}$ of δ. Figure 1 defines a greedy algorithm to argumentatively conduct this search, where: $\textsc{Expand}(\mathit{Fringe}, \gamma)$ removes the configuration $\gamma = (\delta_i, s_i)$ from the search fringe *Fringe*, and replaces it with $\textsc{Moves}(\delta_i, s_i)$; Argue-TAF(*Fringe*, $\mathcal{C}$, $\mathcal{I}$) determines the undefeated conclusions (preference relation $\mathcal{P}$) of a TAF based on the preference arguments that can be constructed amongst pairs of configurations

in *Fringe* using the comparison criteria $\mathcal{C}$ and the ordering over criteria $\mathcal{I}$; and EVAL(*Fringe*, $\mathcal{P}$) selects a configuration γ that is in the optimal subset of *Fringe* based on preference relation $\mathcal{P}$.

This GREEDYARGUE algorithm (Figure 1) does not actually achieve what we desire. Assume that our agent has adopted the Schwartz set criterion [9] as its criteria of optimality. The greediness of the algorithm can lead to sub-optimal final configurations (configurations that are not in the Schwartz set of all final configurations of δ) being selected as a result.

The Greedy Traveling Agent

Consider the following extension of the traveling example of Section 3. Agent a has the following program outlining what to do to travel from point A to point B to point C.

proc travel(A, B, C)
    [drive(A, B) | walk(A, B) | jog(A, B)] ;
    [train(B, C) | skip(B, C)]
endProc

Agent a has the comparison criteria $\mathcal{C} = \{c_1, c_2, c_3\}$ were: c_1 = "*I prefer configurations that promote exercise* (Ex) *to those that demote it*", c_2 = "*I prefer configurations that promote efficiency* (Ef) *to those that demote it*", c_3 = "*I prefer configurations that promote calorie burning* (Ca) *to those that demote it*", and $\mathcal{I}$ expresses that $c_3 >$ (is preferred to) $c_1 > c_2$. Assume agent a believes that: *drive* promotes efficiency, but demotes exercise and calorie burning; *walk* promotes exercise, but demotes efficiency and calorie burning; and *skip* and *jog* together promote calorie burning, while selecting action *train* demotes it.

Assume that $A = home$, $B = uni$, and $C = shop$, and that each of the actions *drive*, *walk*, and *jog* are possible between A and B in situation s_0. The search tree obtained after the first expansion step in the GREEDYARGUE algorithm (Figure 1) is shown in Figure 2.

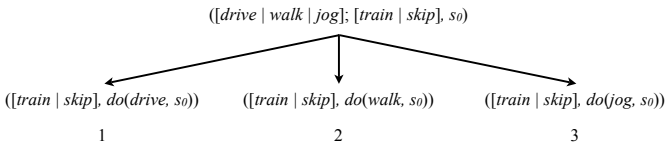

Fig. 2: First EXPAND in the GREEDYARGUE algorithm.

In the ARGUE-TAF step, arguments are constructed in support of $\mathcal{P}(2,1)$ based on exercise, and $\mathcal{P}(1,2)$ based on efficiency. $\mathcal{P}(2,1)$ is the only undefeated conclusion of the *TAF*, resulting in a Schwartz set of $Ch(\{1,2,3\},\mathcal{P}) = \{2,3\}$. If *walk* (node 2) is selected as the first action, the search tree shown in Figure 3 is obtained (assuming both *train* and *skip* are possible after *walk*).

At this stage there are four configurations that may be expanded next, numbered as shown in Figure 3. Arguments may be constructed for: $\mathcal{P}(2,1)$ and

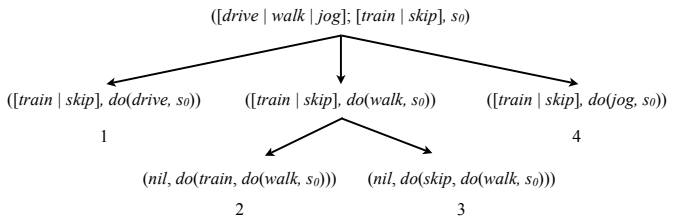

Fig. 3: Second EXPAND in the GREEDYARGUE algorithm.

$\mathcal{P}(3,1)$ based on exercise, and for $\mathcal{P}(1,2)$ and $\mathcal{P}(1,3)$ based on efficiency. The undefeated conclusions of the TAF are $\mathcal{P}(2,1)$ and $\mathcal{P}(3,1)$, resulting in a Schwartz set of $Ch(\{1,2,3,4\},\mathcal{P}) = \{2,3,4\}$. Assume that *train* (node 2) is selected as the last action. As *walk* then *train* represents a final configuration, the search process is complete.

In this example, the resulting configuration is *not* in the Schwartz set of all final configurations of the program given s_0. The *jog* then *skip* combination is preferred to any of the configurations involving *walk*, *drive*, or *train*. Agent a does not examine this path as it cannot form an argument supporting *jog* over *walk* or *drive*, without knowing that after selecting *jog* it can *skip*.

To ensure that only configurations that are in the optimal set of all final configurations of a program are selected as a result, an agent must *speculate* on what actions *could* be performed after transitioning to a configuration, and construct arguments accordingly. An agent speculates as it does not want to search ahead in a program to determine what actions are *actually* possible – this would defeat the purpose of employing informed search.

For each configuration γ_1 on the fringe of a search tree, an agent must find (based on its comparison criteria) all arguments it may form with conclusion $\mathcal{P}(\gamma_1 : \mathbf{a}, \gamma_2)$ where γ_2 is another configuration on the fringe, and $\mathbf{a}$ is a set of actions that can be performed in a *relaxed* interpretation of γ_1. An argument with a conclusion $\mathcal{P}(\gamma_1 : \mathbf{a}, \gamma_2)$ is called a speculative argument, and $\gamma_1 : \mathbf{a}$ is called a speculative extension of γ_1. Informally, an agent determines $\mathbf{a}$ by considering the number of opportunities in γ_1 where an action can be performed, what actions are available to select from at these points, and what actions will allow it to form an argument against γ_2. More formally, $\mathbf{a}$ is defined as a set of actions that are performed in an interpretation of γ_1 using a relaxation of the transition semantics of [3].

A requirement we place upon speculative arguments is that they overestimate what is possible after a configuration is expanded – if there is an actual extension of a configuration γ_1 that is preferred to a configuration γ_2 on the fringe of a search tree, then a speculative argument will be constructed supporting that conclusion. This property is important in proving the soundness of program interpretation by argumentative informed search.

At this preliminary stage, we define the concept of speculation, but the precise mechanism by which a speculation is found (with respect to a general (Con)Golog program) remains unformalised. We imagine that in a relaxation of [3], actions

are always considered as possible. The *Trans* predicate for an action transforms from [3]:

$$Trans(a, s, \delta', s') \equiv Poss(a[s], s) \wedge \delta' = nil \wedge s' = do(a[s], s) \quad \text{to:}$$
$$Trans(a, s, \delta', s') \equiv \delta' = nil \wedge s' = do(a[s], s)$$

Transformation of *Trans* and *Final* predicates in this manner must be applied to each relevant construct (actions, conditionals, while loops, and tests) to formally define a relaxed transition semantics.

Speculative arguments are integrated into a *TAF* (described in Definition 1) as follows. A speculative argument in support of the conclusion $\mathcal{P}(\gamma_1 : \mathbf{a}, \gamma_2)$ is viewed as an argument in support of the conclusion $\mathcal{P}(\gamma_1, \gamma_2)$. Consequently, an argument in support of the conclusion $\mathcal{P}(\gamma_1 : \mathbf{a}, \gamma_2)$ attacks arguments in support of conclusions $\mathcal{P}(\gamma_2, \gamma_1)$ and $\mathcal{P}(\gamma_2 : \mathbf{b}, \gamma_1)$. Defeat amongst arguments is based on the comparison criteria used in their derivation and their relative importance.

Applying speculative argumentation in the extended traveling example allows agent a to construct additional arguments between choices (1) *drive*, (2) *walk*, and (3) *jog*, in the first ARGUE-TAF step shown in Figure 2. Agent a can now construct an argument in support of $\mathcal{P}(3 : \{skip\}, 1)$ and $\mathcal{P}(3 : \{skip\}, 2)$ based on calorie burning. Combining these arguments with those in support of $\mathcal{P}(2, 1)$ and $\mathcal{P}(1, 2)$, based on exercise and efficiency, the undefeated conclusions of the *TAF* are $\mathcal{P}(3 : \{skip\}, 1) \equiv \mathcal{P}(3, 1)$ and $\mathcal{P}(3 : \{skip\}, 2) \equiv \mathcal{P}(3, 2)$. The Schwartz set computed in the first EVAL stage is consequently $Ch(\{1, 2, 3\}, \mathcal{P}) = \{3\}$. Expanding node 3 (selecting *jog* as the first action) results in the search tree shown in Figure 4.

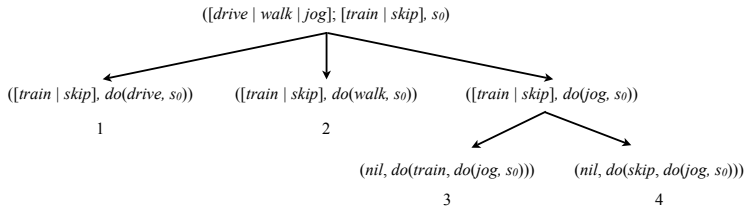

Fig. 4: Second EXPAND step given speculative arguments.

At this stage in the search, agent a constructs arguments in support of $\mathcal{P}(1, 2)$, $\mathcal{P}(2, 1)$, $\mathcal{P}(4, 1)$, $\mathcal{P}(4, 2)$, and $\mathcal{P}(4, 3)$. The only configuration in the Schwartz set of the search fringe is the *jog* then *skip* combination (node 4), which is selected to complete the search process.

False speculations, cyclic preferences, and transient arguments

We now present an example in which sub-optimal final configurations are selected as a result of the algorithm in Figure 1 – even after its modification to include the construction of speculative arguments. This example involves the presence of *transient* arguments. A transient argument is one that supports a preference between two configurations γ_1 and γ_2 on a search fringe, but either (i) does

not support a preference between γ_1 and all extensions of γ_2, or (ii) cannot be used to support preference between all extensions of γ_1 and the configuration γ_2. Arguments based on false speculations (impossible speculative extensions) are an example of transient arguments. We assume in this example that the Schwartz set [9] has been adopted as the optimality criterion.

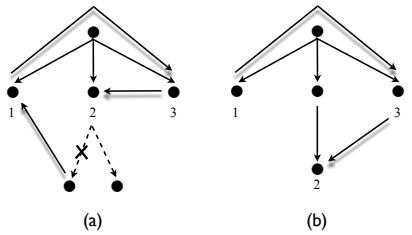

Fig. 5: An impossible speculative extension in a cycle of preferences.

Consider the case shown in Figure 5(a), where the fringe of a search tree has three choices. A speculative extension of choice 2 is preferred to choice 1, resulting in a cycle of preference relationships amongst nodes on the search fringe. In this case, the Schwartz set consists of all three choices and choice 3 may be selected for expansion to complete the search process. Assume the speculative extension is not actually possible – Figure 5(b) shows all final configurations of the program in this case and the preferences between them. Choice 3 in Figures 5(a) and 5(b) is *not* in the Schwartz set of all final configurations.

We expect the presence of transient arguments to impact the result of search in different ways given the adoption of different optimality criteria. Based on the example present above, we can arrive at the following general conclusion. The impact of transient arguments, irrespective of the optimality criterion adopted, is that they can result in situations where a reliable decision regarding the optimality of a final configuration cannot be made.

In Section 5, a method of prioritised node expansion is developed which, assuming a set of conditions are satisfied by the adopted optimality criterion, prevents an agent from selecting a final configuration if the search fringe is in a state that does not admit a reliable decision. If the search fringe is in an unreliable state, the agent must explore a non-final configuration in the hope of transitioning to a fringe that admits a decision. In outlining conditions that must be satisfied by all optimality criteria that may be adopted by an agent, we are able to define the situations in which a reliable decision can be made, irrespective of which criterion is chosen.

5 Prioritised node expansion

To address problems arising from transient arguments, prioritised node expansion is incorporated into the algorithm presented in Figure 1, in conjunction with the construction of speculative arguments. Prioritisation must be able to guide the agent away from situations in which it cannot make a clear decision (according to its criterion) toward situations in which it can. To facilitate this,

we make the following assumption regarding the nature of all optimality criteria an agent may adopt.

Assumption 1 *Given a non-empty set of choices $\mathcal{A}$, and a preference relation over those choices $\mathcal{P}$, any optimality criterion adopted by an agent finds a non-empty subset $Ch(\mathcal{A}, \mathcal{P}) \subseteq \mathcal{A}$ where $Ch(\mathcal{A}, \mathcal{P})$ contains (at least) all choices $\gamma \in \mathcal{A}$ for which there are no choices $\gamma' \in \mathcal{A}$ where $\mathcal{P}(\gamma', \gamma)$. In any optimality criterion adopted by an agent, the question of whether or not a choice $\gamma \in \mathcal{A}$ is optimal is independent of any preference relationship in $\mathcal{P}$ that does not form part of an undirected path in $\mathcal{P}$ between γ and another choice $\gamma' \in \mathcal{A}$.*

Assumption 1 expresses that if an agent cannot find a choice better than γ, γ must be optimal, and the question of whether or not a choice is optimal is independent of all choices not connected to it in $\mathcal{P}$.

The prioritisation algorithm

Given a search fringe *Fringe* and a preference relation $\mathcal{P}$ over the configurations in *Fringe*, EVAL(*Fringe*, $\mathcal{P}$) in Figure 1 is redefined to:

(i) Compute the optimal subset $Ch(Fringe, \mathcal{P})$ of configurations in *Fringe* using an adopted optimality criteria;
(ii) Select a configuration $\gamma \in Ch(Fringe, \mathcal{P})$ for which there is no configuration $\gamma' \in Fringe$ such that $\mathcal{P}(\gamma', \gamma)$, if one exists;
(iii) If (ii) cannot be satisfied, select a configuration $\gamma \in Ch(Fringe, \mathcal{P})$ which is not connected by an undirected path of preferences in $\mathcal{P}$ to a non-final configuration $\gamma' \in Fringe$, if one exists;
(iv) If (ii) and (iii) cannot be satisfied, select a non-final configuration $\gamma \in Ch(Fringe, \mathcal{P})$, if one exists;
(v) If (ii), (iii), and (iv) cannot be satisfied, select any non-final configuration $\gamma' \in Fringe$.

Giving preference to non-final configurations (iii, iv, v) increases the amount of state-space searched, but is necessary to encourage search into a state where clear decisions regarding optimality can be made.

Theorem 1. *Given an optimality criterion $\mathcal{O}$, satisfying Assumption 1, any final configuration of a program δ, given an initial situation s_0, found as a result of argumentative informed search with prioritisation is in the optimal subset of all final configurations of δ determined by $\mathcal{O}$.*

Proof. We demonstrate that a final configuration γ of δ cannot be selected from any search fringe *Fringe*, given a preference relation amongst its members $\mathcal{P}$, if it is not in the optimal subset of $\mathcal{A}$ according to $\mathcal{O}$. Based on the redefinition of EVAL according to the prioritisation algorithm, $\gamma \in Ch(Fringe, \mathcal{P})$ will only be selected as a result if: (i) there is no configuration $\gamma' \in Fringe$ such that $\mathcal{P}(\gamma', \gamma)$; 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 non-speculative or speculative argument that can be formed in support of a configuration $\gamma' \in \mathit{Fringe}$ being preferred to γ. As speculation overestimates what is possible after expanding a configuration, there is therefore no final configuration of δ given s_0 that is preferred to γ. According to Assumption 1, γ must be a member of the optimal subset of $\mathcal{A}$ according to $\mathcal{O}$.

If (ii) holds, the set of configurations connected to γ by an undirected path in $\mathcal{P}$, U, are final. As γ is not connected to a non-final configuration by an undirected path in $\mathcal{P}$, no configuration in U is connected to a non-final configuration by an undirected path in $\mathcal{P}$. Moreover, no speculative argument can be formed to connect γ to a non-final configuration in this way. As speculation overestimates what is possible after expanding a configuration, there are no final configurations of δ outside of U that are connected to γ (or any configuration in U) by an undirected path of preferences. According to Assumption 1, the question of whether γ is optimal is independent of any configuration in $\mathcal{A}$ outside of U. As $U \subseteq \mathcal{A}$, and γ is optimal w.r.t U according to $\mathcal{O}$, γ must be optimal w.r.t $\mathcal{A}$ according to $\mathcal{O}$.

6 Future Work – Single to Multi-Agent

In a multi-agent system, each agent has a set of programs outlining the tasks it needs to perform to achieve its goals. In ConGolog, a set of programs may be executed concurrently, where at each step multiple actions (one from each program) may be performed. In this section, we discuss how the single-agent search for an optimal execution of a Golog program, presented in this paper, may be generalised to a multi-agent setting and applied to ConGolog programs.

The search process in a multi-agent setting may operate in much the same way as in the single agent case. Agents will start with an initial configuration (δ, s_0), where δ represents a collection of programs to be executed concurrently. The legal moves function $\textsc{Moves}(\delta, s_0)$ determines which transitions may be made from (δ, s_0) based on what actions can be performed at the same time. Each agent will have a set of criteria for comparing configurations. Using these criteria, each agent can construct arguments supporting preference over unexpanded nodes, as in the single-agent case. A multi-agent version of the tournament-based argumentation framework (TAF) must be defined to determine what forms of defeat are meaningful (and possible) amongst preference arguments formed by different agents. These agents may have different comparison criteria, and different importance orderings over those criteria.

In a multi-agent setting, agents may have disagreements regarding what holds in any given situation, based on different and inconsistent beliefs concerning the initial situation. Not only does the ability to incorporate arguments based on fact in a TAF become important, but the ability to handle dispute over what actions are possible in the $\textsc{Moves}$ function becomes necessary.

As voting criteria are designed to take the preferences of multiple parties and select an optimal subset of a choice set, its use in the approach presented in this paper provides a good platform for extension to a multi-agent domain.

In addition to extending the presented approach for use in a multi-agent setting, we believe further investigation is worthwhile on: (i) the complexity of forming speculative arguments, (ii) the impact of the comparison criteria adopted by an agent on the number of speculative arguments formed, and (iii) the use of constraints on, or heuristics for, the generation of speculative arguments to focus only those that have an impact on which paths are chosen.

7 Conclusion

In this paper, an argumentative informed search method for interpreting Golog programs was described. The approach is based on the use of argumentation to allow flexible comparison between final and non-final configurations of a program, and the use of informed search to reduce the space of configurations to be considered in finding an optimal result. Arguments are constructed and analysed to form a preference relation amongst unexpanded nodes in a search tree, the paths of which represent executions of a Golog program. A voting criteron, such as the Schwartz set [9], is used to select nodes within the search fringe that are acceptable for expansion. Speculative arguments are used to predict future opportunities at each stage of the search, and prioritised node expansion is incorporated to ensure that only optimal executions are found as a result.

References

1. T. J. M. Bench-Capon. Persuasion in practical argument using value based argumentation frameworks. *Journal of Logic and Computation*, 13:429–48, 2003.
2. C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In *AAAI*, pages 355–362, 2000.
3. G. de Giacomo, Y. Lesperance, and H. J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1-2):109–169, 2000.
4. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming, and n-person games. *Artificial Intelligence*, 77:321–357, 1995.
5. C. Fritz and S. McIlraith. Compiling qualitative preferences into decision-theoretic Golog programs. In *NMR Workshop at IJCAI*, 2005.
6. Y. Lespérance, H. J. Levesque, and R. Reiter. A situation calculus approach to modeling and programming agents. *Foundations and Theories of Rational Agency*, pages 275–299, 1999.
7. H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–83, 1997.
8. J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.
9. T. Schwartz. Rationality and the myth of the maximum. *Noûs*, 6(2):97–117, 1972.
10. E. Shafir, I. Simonson, and A. Tversky. Reason-based choice. *Cognition*, 49:11–36, 1991.
11. C. Starmer. Explaining risky choices without assuming preferences. *Social Choice and Welfare*, 13(2):201–213, 1996.