Efficient storage and retrieval in protocol libraries using subsumption hierarchies

Tim Miller[1] and Peter McBurney[2]

[1] Department of Computer Science and Software Engineering
University of Melbourne, Victoria, 3010, Australia
tmiller@unimelb.edu.au
[2] Department of Computer Science, University of Liverpool, Liverpool, L69 7ZF, UK
mcburney@liverpool.ac.uk

Abstract. For an agent to intelligently use specifications of executable protocols, it is necessary that the agent can quickly and correctly locate a protocol that achieves its goals. Techniques such as model checking or theorem proving can be used to assess whether a protocol achieves a goal; however, for resource bound agents, this approach may be inefficient. Building on previous work on characterising and matching protocols, we present a method for structuring and searching protocol libraries using subsumption hierarchies. These hierarchies are directed graphs, in which the vertices are characterisations of protocols, and edges record a relation between two characterisations if one characterisation subsumes the other; that is, the characterised protocol achieves all of the same outcomes. An experimental analysis demonstrates that this approach is more efficient for anything other than the smallest of protocol libraries.

1 Introduction

In the distributed environments of multi-agent systems, interaction protocols are seen as a promising approach to coordination in multi-agent systems. However, many practitioners view interaction protocols as rigid specifications that are defined *a priori*, with agents being hard-coded to follow the protocol rules. We identify three significant disadvantages with this approach:

1. it strongly couples agents with the protocols they use — something which is unanimously discouraged in software engineering — therefore requiring agent code be changed with every change in a protocol;
2. agents can only interact using protocols that are known at design time, a restriction that seems out of place with the goals of agents being intelligent and adaptive; and
3. agents cannot compose protocols at runtime to bring about more complex interactions, therefore restricting them to protocols that have been specified by human designers — again, this seems out of place with the goals of agents being intelligent and adaptive.

An important corollary of these points is that the protocol is internalised within the individual agents. There is no possibility to communicate, inspect or verify the protocols by the agent or others, not to mention a duplication of effort for each agent engineer as each agent must be encoded with the same protocol. As a result, we believe it is important that agent interaction protocols are first-class computational entities, allowing agents to select, reference, share, compose, invoke and inspect protocols at runtime. Such an approach would allow agents to assess which protocols achieve their goals, and to learn the rules and effect of new protocols at runtime. We call such protocols *first-class protocols* [5].

A major goal of research into first-class protocols is for agents to maintain a library of interaction protocols, and to be able to select the protocol that best suits the goals that it wants to achieve at a given time and in a given environment. For this, agents must be able to quickly and correctly determine the outcomes that can result for an interaction protocol, and compare protocols in their library.

In previous work [6, 8], we presented methods to *characterise* the possible outcomes of a first-class protocol, so that an agent does not have to calculate the outcomes each time it is trying to find a suitable protocol, and to *match* a protocol that achieves a goal against these characterisations. In that work, we assume that agents have access to a library of protocols, but we made no assumptions about the structure of such libraries, nor how the libraries are searched. The characterisation and matching method are summarised in Section 3.

In this paper, we improve that method by outlining a method for structuring and searching a library of protocols using *subsumption hierarchies* [11] — directed graphs that record the relationship between characterisations. By structuring the characterisations as a subsumption hierarchy, rather than a list, we take advantage of the subsumption relationship to reduce the search space. In Section 4.3, we present an informed depth-first search algorithm for locating all matching protocols in a hierarchy, and in Section 5, we present an experiment comparing the complexity of the search algorithm in comparison to an uninformed linear search on an unstructured library. Emphasis is placed on protocols specified in the $\mathcal{RASA}$ protocol language [5], discussed in Section 2, which uses constraint languages to represent messages and meaning, but we believe such ideas would be applicable to other protocol languages that use propositional logic. The general approach should also be applicable to other aspects of multi-agent systems, such as plan libraries.

2 The $\mathcal{RASA}$ Framework

The $\mathcal{RASA}$ specification language was designed as an example of the minimal operators that would be required for a successful first-class protocol specification language. First presented in [5], along with its operational semantics, the language uses constraint languages and process algebra to specify interaction protocols. In this section, we briefly present this language, and a logic for reasoning about protocols specified in this language.

2.1 Modelling Information

We assume that agents communicate about their universe of discourse using some constraint language. Rather than enforce a particular language, the $\mathcal{RASA}$ framework simply assumes that the communication language fits the definition of a *cylindric constraint system* proposed by De Boer *et al.* [1], which is that a constraint language is a *complete algebraic lattice*, with an operator for hiding variables. That is, a structure $\langle C, \sqsupseteq, \sqcup, \text{true}, \text{false}\rangle$, in which C is the set of constraints, $\sqsupseteq$ a partial order between elements of C, $\sqcup$ is the least upper bound (or join) operator, and true and false are the least and greatest elements of C respectively. $\sqsupseteq$ is an entailment operator, such that $c \sqsupseteq d$ means that the information in d can be derived from c, for example $x = 1 \sqsupseteq x > 0$.

A constraint is one of the following: an atomic proposition, c, for example, $X = 1$, where X is a variable; or a conjunction, $\phi \sqcup \psi$, where ϕ and ψ are constraints. We extend this notation by allowing negation on the right of an entailment operator, for example, $\phi \sqsupseteq \neg\psi$ is true if and only if $\phi \sqsupseteq \psi$ is not. Other propositional operators such as disjunction and implication are then defined from these. We will continue to use the meta-variables ϕ and ψ to refer to constraints throughout this paper. We also use $vars(\phi)$ to refer to the free variables that occur in ϕ.

2.2 Modelling Protocols

The $\mathcal{RASA}$ protocol specification language is a simple action language, in which the actions are messages sent over a channel. We use the notion of *state* in the language. State is useful, because it allows us to build up the meaning of protocols compositionally, for example, the effect of sending two messages is the effect of sending the second message in the state that results after sending the first. The final outcome of the protocol is the end state. A detailed presentation of the specification language, including operational semantics, is available in [5].

The details of the language are not particularly relevant to this paper, but we give a brief overview for interest. Atomic protocols are specified as triples: a precondition, a message template, and a postcondition. We write this as $\psi \xrightarrow{\phi_m} \psi'$. If a precondition holds in the current state, a message that fits the template can be sent, and the result of sending the message is the updating of the state from the postcondition. Compound protocols can be built-up using protocol operators; the two most important are sequential composition and choice. For two protocols, α and β, their sequential composition is defined as $\alpha; \beta$, and their choice composition is defined as $\alpha \cup \beta$. The empty protocol is defined as $\psi \rightarrow \epsilon$, in which ψ represents the precondition that must hold for the empty protocol to be enabled.

A *protocol specification* is a collection of *named* protocol definitions of the format $N \mathrel{\widehat{=}} \alpha$, in which N is the name of the protocol α. Protocol definitions can reference each other via their names, making recursive definitions possible. Using this, we can define iteration, α^*, which represents iterating over α zero or more times, as the protocol N, in which N is defined as $N \mathrel{\widehat{=}} \epsilon \cup \alpha; N$.

2.3 Reasoning about Protocols

$\mathcal{RASA}$ defines a logic for reasoning about protocols. The logic is concerned with protocol outcomes; that is, the state of the protocol after it is executed. For this reason, we have adapted a version of propositional dynamic logic [3] to tailor it to the $\mathcal{RASA}$ specification language, and derived a proof system that corresponds to the system for dynamic logic.

The syntax for a proposition in this logic is defined using the following grammar, assuming that ϕ_0 is a constraint in the underlying constraint language:

$$\phi \ ::= \ \phi_0 \ \mid \ \phi \wedge \phi \ \mid \ \neg\phi \ \mid \ [\alpha]\phi$$

A formal semantics for this logic has been defined in [9]. Each proposition is evaluated under a model — a constraint representing a possible state of the system. If ψ_0 is this state, then ϕ_0 is true if and only if $\psi_0 \sqsupseteq \phi_0$. $\phi \wedge \psi$ and $\neg\phi$ are defined as conjunction and negation respectively. The interesting operator, $[\alpha]\phi$, which is found in propositional dynamic logic, has the meaning that ϕ holds for every possible outcome state of the protocol α. That is, no matter which interaction path is taken in the protocol α, the proposition ϕ will hold after the protocol has executed. We also define another operator from dynamic logic: $\langle\alpha\rangle\phi$, which is the dual of $[\alpha]\phi$, and means that ϕ holds in at least one possible outcome of the protocol α. This is defined as shorthand for $\neg[\alpha]\neg\phi$.

We use subscripts on the Greek letters ϕ and ψ to indicate something that is strictly a constraint; that is, ϕ_0 is a constraint, while ϕ can be a constraint or a dynamic logic proposition.

3 Characterising and Matching Protocols

Characterisations for a protocol are derivable directly from the protocol specification itself. In this section, we discuss an algorithm for characterising protocols, including iterative protocols, and terminating recursive protocols; that is, recursively defined protocols that always terminate. We then discuss how these characterisations can be used to match which protocols achieve a specified goal.

3.1 Definitions

Before we present previous work on characterisation and matching of protocols, we briefly define some terms.

The *weakest precondition* of a protocol is the weakest (or most general) proposition from which a protocol will execute and terminate. The *maximal postcondition* is the strongest proposition that results from a protocol being executed under its weakest precondition.

A *goal* is a state of the world that an agent would like to bring about, or maintain. In this paper, we assume that a goal is represented as a proposition in the underlying constraint language.

Given a goal, ϕ_G, and an initial state, ψ_I (the state of the world from which an agent wants to achieve the goal – generally the current state), a *weak matching* protocol is a protocol, α, that achieves the goal ϕ_G from the initial state ψ_I for at least one outcome. Formally:

$$\psi_I \rightarrow \langle\alpha\rangle\phi_G$$

A *strong matching* protocol is a protocol that achieves a goal for all outcomes, assuming that there exists at least one outcome[3]. Formally:

$$\psi_I \rightarrow [\alpha]\phi_G.$$

All strong matching protocols are also weak matching protocols. We distinguish between the two because an agent would want a protocol that achieves its goal for at least one outcome, but would likely prefer a protocol that achieves it for all outcomes.

To find all matches for a goal ϕ_G from the state ψ_I, the agent could simply use the PDL proof system discussed in Section 2.3. That is, for every protocol α, if the proof $\psi_I \rightarrow \langle\alpha\rangle\phi_G$ is successful, then α is a weak match. For a large protocol library, this is an expensive operation to perform each time an agent wants to find a protocol that achieves a certain goal. Instead, we summarise the preconditions and outcomes of the protocol using characterisations, and then search these.

3.2 Representing Characterisations

Characterisations are represented as theorems in the logic presented in Section 2.3. For example, the characterisation

$$\psi_0 \rightarrow [\alpha]\phi_0$$

specifies that, if executed from any state that satisfies the weakest precondition ψ_0, the protocol α is guaranteed to achieve the outcome ϕ_0. Our goal is to characterise, for each protocol in a protocol library, not the outcomes that it achieves, but the outcomes of the *paths* of the protocol.

As an example, consider the protocol $A;(B \cup C)$, in which $P \mathrel{\widehat{=}} p \xrightarrow{p} p'$, $Q \mathrel{\widehat{=}} \text{true} \xrightarrow{q} q'$, and $R \mathrel{\widehat{=}} \text{true} \xrightarrow{r} r'$. Figure 1 shows the abstract syntax tree of the protocol, and the characterisations that we want to be able to derive. This protocol contains two paths: (1) P followed by Q; and (2) P followed by R.

To demonstrate why we characterise paths, rather then entire protocols, consider the protocol $P \cup Q$. Our algorithm will generate two characterisations: $p \rightarrow [P]p'$ and $q \rightarrow [Q]q'$. If we were to characterise the outcomes of $P \cup Q$ in one characterisation, we could write $p \vee q \rightarrow [P \cup Q](p' \vee q')$. However, this loses vital information: that of the relationship between p and p', and the relationship between q and q'. That is, p' is only achievable from a state in which p holds. One can not infer this from the general characterisation, therefore, an agent could conclude that p' may achievable from q, which is not the case.

[3] This assumption is subsumed by our assumption that a protocol is free from stuckness.

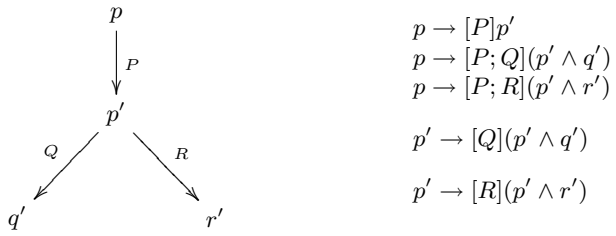

Fig. 1. An abstract syntax tree for a protocol, and its characterisations.

3.3 Characterising Protocols

Recall from Section 3.2, that protocols are characterised by the outcomes of their paths. To do this, each path in the protocol is symbolically executed, with the initial state being its weakest precondition. When the algorithm reaches the end of a path, the symbolic state that is left is the maximal postcondition of that path. The characterisation can be derived from this maximal postcondition, the weakest precondition, and the path.

In previous work [8], we present a symbolic execution algorithm for producing characterisations of protocols, including infinitely iterative and finite recursive protocols. We prove that, for a restricted class of infinitely recursive protocols, characterisation can be used by re-writing the protocol into an equivalent iterative protocol.

3.4 Matching Characterisations

To prove that a protocol, α, is a strong match for goal ϕ_G and initial state ψ_I, one must prove the following:

$$\forall a \in chrs(\alpha) \bullet \texttt{strong\_match}(a, \psi_I, \phi_G)$$

in which `strong_match` is defined as the relation

$$\texttt{strong\_match}(\psi_0 \rightarrow [\alpha]\phi_0,\ \psi_I,\ \phi_G) \iff \psi_I \sqsupseteq \psi_0 \ \rightarrow\ \phi_0 \sqsupseteq \phi_G$$

and $chrs$ returns the set of all characterisations for a protocol. The above states that, given a set of outcome characterisations for a protocol, if for all of these outcomes, when the initial state satisfies the precondition, the postcondition satisfies the goal state, then we have a strong match.

Recall from Section 2.3 that $\langle\alpha\rangle\phi$ is defined as shorthand for $\neg[\alpha]\neg\phi$. Using this symmetry, weak matching is defined as:

$$\neg\forall a \in chrs(\alpha) \bullet \texttt{strong\_match}(a, \psi_I, \neg\phi_G).$$

Previous work [8] shows that this characterisation and matching method is sound and complete.

4 Structuring and searching protocol libraries

In this section, we present our method for structuring protocol libraries for efficient searching. This method takes advantage of the partial ordering of constraint systems; that is, it relies on the entailment operator, $\supseteq$, and that protocol outcomes may often be related via this partial order.

4.1 Protocol characterisations and posets

The definition from Section 2.1 of the underlying constraint language notes that our constraint system satisfies the properties of a lattice. We note that, as a result of this, our constraint system also satisfies the properties of a *partially-ordered set* (poset), a set paired with a partial order; in this case, the poset is $(C, \supseteq)$.

In this work, we are interested in structuring protocol outcomes. From Section 3, we know that protocol outcomes are specified using constraints; that is, for any characterisation, $[\alpha]\phi_0$, of protocol α, ϕ_0 is a constraint.

From the theory of posets, we know that, given a poset $(S, \leq)$, then, for any set, S_0, such that $S_0 \subset$ S, $(S_0, \leq)$ is also a poset. From this, we make the following observation:

> The set of all possible characterisations, $\mathcal{A}$, forms a poset with the operator $\leq$, $(\mathcal{A}, \leq)$, in which the relation $[\alpha]\phi_0 \leq [\beta]\psi_0$ holds if and only if $\psi_0 \supseteq \phi_0$. Any library of characterisations is a subset of $\mathcal{A}$, therefore, any library forms a poset with the inverse of the entailment operator.

This observation means that, given the set of protocols available to an agent, and their characterisations, we can form a poset from these characterisations by using the entailment operator as the partial order, ignoring the protocol that is paired with a characterisation.

4.2 Protocol libraries as subsumption hierarchies

A *subsumption hierarchy* is a directed, acyclic graph (DAG), in which the nodes of the graph represent values, and the relationships between nodes represent that the source node subsumes the destination node. Given a poset, $(S, \leq)$, one can represent that poset in a subsumption hierarchy by taking S as the nodes of the graph, a creating an edge, (s, t), between the two nodes, s and t, if and only if $s \leq t$. Thus, we can represent the poset $(\mathcal{A}, \leq)$ using a subsumption hierarchy.

Example 1. Consider an example of a collection of outcomes, which express constraints over the variables x, y, and z using a simple constraint language with membership and equality operators. Characterisations represent nodes, and, using the entailment operator as the partial order, one creates the DAG of characterisations shown in Figure 2. In this figure, nine protocols are displayed, represented by the capital roman letters *A, B, ..., I.*

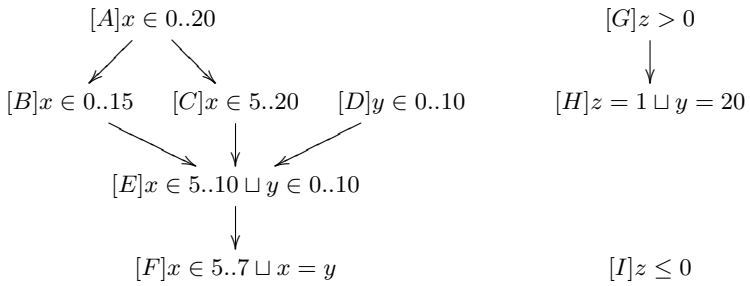

Fig. 2. An example library organised as a subsumption hierarchy using the entailment operator as a partial order.

It is straightforward to see that the constraint at each node subsumes the constraints at all nodes below it. Note that it is not the case that every graph is connected to every other graph. In this example, there are three distinct subgraphs.

There exist several efficient algorithms for creating subsumption from posets; however, it would be wise for an agent to implement an incremental algorithm. That is, an algorithm that can take one element, and insert it into the correct position of an already-constructed graph. This is essentially a search, because it is finding the correct position to insert the node, and could be achieved using many different algorithms, such as depth-first or breadth-first searches.

Structuring a protocol library as a subsumption hierarchy allows agents to perform an informed search over the library, rather than having to search through every protocol as it must do in a unstructured library. We explore this further in the following section.

4.3 Searching Structured Libraries

In this section, we present in outline an algorithm for searching libraries structured as subsumption hierarchies, as discussed in Section 4.2. In Section 5, we discuss the complexity of such an algorithm, comparing this to searching an unstructured library.

Unstructured Libraries If an agent's library is unstructured, and it wants to find all protocols that achieved a certain goal, it would have try matching every characterisation in the library. We assert that, if an agent structures its library as outlined in Section 4.2, this can be improved upon.

Structured Libraries We first note the following two theorems.

Theorem 1. *If* $([\alpha]\phi_0, [\beta]\psi_0)$ *is an edge in a subsumption hierarchy, and execution of protocol* α *achieves a goal, then execution of protocol* β *will also achieve that goal.*

Proof. From the edge $([\alpha]\phi_0, [\beta]\psi_0)$, we know that $\psi_0 \sqsupseteq \phi_0$. If α achieves a goal, ϕ_G, then it must be that $\phi_0 \sqsupseteq \phi_G$. From the transitive of the entailment operator, we know that $\psi_0 \sqsupseteq \phi_G$, and therefore, $[\beta]\psi_G$ holds. □

Theorem 2. *If* $([\alpha]\phi_0, [\beta]\psi_0)$ *is an edge in a subsumption hierarchy, and all outcomes of execution of protocol* α *are inconsistent with a goal* ϕ_G *— that is,* $\phi_G \sqcup \phi_0 \sqsupseteq$ *false —, then all outcomes of execution of protocol* β *will also be inconsistent with that goal.*

Proof. To prove this, we take the following property of lattices: if $\phi_0 \sqsupseteq$ false, then $\phi_0 \sqcup \psi_0 \sqsupseteq$ false. That is, if ϕ_0 entails false, then adding more information to ϕ_0 will never make it true. Now, if α is inconsistent with the goal ϕ_G, we know that $[\alpha]\neg(\phi_0 \sqcup \phi_G)$. The constraint ψ_0 entails ϕ_0 (from $[\alpha]\phi_0 \leq [\beta]\psi_0$), so ψ_0 is equivalent the constraint $\phi_0 \sqcup \phi_1$, for some ϕ_1. From this and the property of lattices discussed above, we know that $[\beta]\neg(\psi_0 \sqcup \phi_G)$, so β is inconsistent with the goal. □

From these theorems, we know that, given a subsumption hierarchy, the following two properties hold:

1. If an agent is attempting to match a goal, ϕ_G, and it evaluates a node, ϕ_A, such that $\phi_A \sqsupseteq \phi_G$, then every node that is a transitive child of ϕ_A in the graph must also satisfy the goal.
2. If an agent is attempting to match a goal, ϕ_G, and it evaluates a node, ϕ_A, such that $\phi_A \sqcup \phi_G \sqsupseteq$ false — that is, ϕ_A and ϕ_G are inconsistent with each other — then every transitive child of ϕ_A must also be inconsistent with ϕ_G

This can cut our search time down quite effectively, depending on the graph and the goal. Consider the graph from Figure 2, and the goal $x \in 0..30$. The agent matches the node $x \in 0..20$, because this entails the goal, the agents knows that every protocol at that node and at all four nodes transitively related to it, satisfy the goal, and are implicitly matched. Alternatively, if the goal is $x = 50$, then the constraint $x \in 0..20 \sqcup x = 50$ is unsatisfiable, so there is no need to search the children of the node $x \in 0..20$, because none of them will be consistent with $x = 50$ either.

Depth-First Search If we were to search a structured library using a depth-first search, we can use the above properties to reduce the search. Given a goal, the search continues deepening until one of four properties holds for a node:

1. the node has no children (a dead end);
2. each of the nodes children has been visited already;
3. the characterisation on the node implies the goal; or
4. the characterisation and the goal are incompatible.

The first and second properties are standard for a depth-first search, and in those instances, the algorithm retreats one edge to node from which it came, and continues. However, the third and fourth properties remove any transitive children from the search. In the third case, all nodes subsequent in the search are added as matches, and in the fourth case, no nodes are added. As with the first two cases, the algorithm retreats to the parent and continues.

4.4 Improving search efficiency using variable propagation

One downfall of this method is that an entire hierarchy may be searched despite their being no chance of a match. Consider the example graph in Figure 2. The largest subgraph in this graph contains a source node representing a protocol labeled A with the characterisation $[A]x \in 0..20$, which has a collection of (transitive) descendents, each of which further constrains the value of x, or the values of x and y. Now, consider that an agent has a goal $z = 0$. When evaluating the source node, the agent will note that $x \in 0..20$ and $z = 0$ are consistent with each other, but that $x \in 0..20$ does not achieve the goal $z = 0$. As such, it will continue deepening its search until the entire graph has been traversed, finding no matches.

In Figure 2, each of the nodes in the left-most hierarchy (i.e., that sub-graph containing the six protocols labeled A through F) refer only to the variables x or y, therefore, none can constrain the value of z. If we consider an agent with a large protocol library, it is likely that for an arbitrary goal, most of the protocols will not refer to variables in that goal. Using the search process outlined above will result in the agent assessing most protocols in the library — an approach which is not much more efficient than simply assessing every protocol.

To improve this, we alter both our process of structuring libraries and searching them. Structuring is altered by recording the set of free variables in a characterisation, as well as all free variables in all descendents. In Figure 2, all nodes in the left-most hierarchy would be paired with the set of variables x, y, the unconnected node would be paired with z, while the final two nodes would have y, z. We refer to this as *variable propagation*, from the fact that the variables propagate up the graph.

We add an additional condition for terminating the deepening of our depth-first search algorithm: the cases in which all variables in the goal are not annotated to the current node. In our above example, if an agent has a goal containing only the variable z, then it calculates that $\{z\} \not\subseteq \{x, y\}$, therefore eliminating the largest graph in Figure 2 from our search. Assuming that comparing the free variables in characterisations is more efficient than the entailment operator in our constraint language, which we do not believe is unreasonable, this can reduce the overall complexity of our search.

4.5 Termination, Soundness, and Completeness

Theorem 3. *The depth-first search algorithm terminates.*

Proof. Termination of this improved algorithm is straightforward to show. As with any depth-first search, deepening terminates when they are no children at the current node, which is guaranteed to terminate because the hierarchy is both finite and acyclic. The additional conditions for termination cannot cause the algorithm to recurse indefinitely. □

Theorem 4. *The depth-first search algorithm is sound, and is complete for goals containing no tautologies.*

Proof. The method for matching a single protocol has been proved sound in earlier work [8]. For the depth-first search, the algorithm includes all of the current node's children as matches as well. This is sound from Proposition 1.

For completeness, we have to demonstrate that those nodes in the graph that are omitted from the search cannot possibly achieve the goal. However, this is not the case, but it is *almost* the case.

Firstly, we identify a counter-example showing that our algorithm is not complete. Consider a characterisation $[N]x = 1$, and a goal $x = 1 \sqcup y = y$. Clearly, the proposition $[N](x = 1 \sqcup y = y)$ holds, because $x = 1$ holds for all end states of N, and $y = y$ is a tautology. However, if the characterisation is in a graph containing on the variables x, the proposition $[N](x = 1 \sqcup y = y)$ will never be proved, because the deepening will terminate at the source node, for which the propagated set of variables, $\{x\}$, is not a superset of the variables in the goal.

If we assume that agents do not contain tautologies on variables in their goals, then our algorithm is complete. To prove this, we must show that characterisations that omitted from the search cannot possibly achieve the goal. There are two cases in which characterisations are omitted: if the goal and current node are incompatible, and if the variables in the goal are not in the propagated variables at the current node. Theorem 2 shows that the first case is correct. For the second case, if we consider the characterisation, $\langle\alpha\rangle\phi_A$, of any arbitrary child of the current node, then it is not possible that $\langle\alpha\rangle\phi_G$, for goal ϕ_G. We know that the variables in the goal are not a subset of the propagated variables at the current node. All children of the current node have at most, the same variables as the current node. The definition of $\supseteq$ specifies that an entailment is true if and only if the possible bindings for all variables on the left is a subset of the possible bindings for all variables on the right. If there is a variable on the right that is not on the left, then the possible binding on the left includes all bindings for that variable, therefore, the only time this could be entailed is when the possible bindings on the right contains all bindings. From this, we conclude that constraints on that variable must be a tautology. Since we assume that there are no tautologies on variables, then our algorithm is complete. □

Regarding the issue of tautologies in goals, we offer two items of discussion. Firstly, the algorithm can be modified to check for tautologies; that is, prove that the constraints on variables not in the protocol are universally true. Alternatively, the algorithm can be left as is, because we believe the approximate solution is better than the above alternative. It would be unusual for a rational

agent to contain a tautology in its goals — in fact, we believe this would indicate an agent with far greater problems than matching protocols — and testing for tautologies in a goal would be unnecessarily increasing the overhead of the search, just to avoid a problem that is unlikely to occur.

5 Analysis

In this section, we present an analysis of the structuring and searching of protocol libraries, and compare this to the case of unstructured libraries. We focus on the number of entailments that the constraint system will have to perform, because we believe that this would be the most expensive part of the search process. Our assertion is that using structured libraries significantly reduces the number of entailments.

5.1 Informal analysis

We assume that an unstructured library would be implemented using a list-like structure. For such an implementation, the insertion of a new protocol into the library would have time complexity $O(1)$: the new protocol is simply added to the end of the list.

For search, we assume that an agent would like to find every possible match in its protocol libraries, and then deliberate over these further. Such a search requires an agent to check every characterisation, each requiring two entailments (one for the precondition and one for the postcondition), so for an unstructured library, the best and worst cases are both in $O(n)$.

Assuming a structured library and using the algorithm presented in Section 4.3, an agent can significantly improve on its search time. The downfall to our method is that insertion time is increased. Inserting a node would require an algorithm that searches each connected graph until either a correct place is found, or until the graph has been completely traversed. Assuming that the agent uses a standard depth-first search without analysing the variables at nodes, the complexity of the insertion is $O(n)$.

Searching the libraries is where we see an improvement. The best and worst-case complexities are straightforward: the best case is that every protocol in the library forms a single connected graph with exactly one source node, and the goal is entailed by that source node. This has the complexity of $O(1)$. The worst case is the same as for a linear search: when the goal is entailed by no characterisations, but compatible with all, and is therefore $O(n)$.

For the average-case, we have a number of considerations. Firstly, the structure of the graphs can range anywhere from those in the best-case analysis to those in the worst-case analysis. At present, we have little idea as to the relationships between protocol outcomes, or the probability of how many protocols will be entailed by others. In addition, these properties will vary depending on the constraint language that is used, the domain of the protocols, the types of the protocols, and the nature of the interactions the protocols define. Therefore,

instead of a formal analysis, we performed an experimental analysis, which gave us some idea of the complexity of our depth-first search approach.

5.2 Experimental Evaluation of Average-Case Complexity

For the purpose of experimentation, we use an integer-based constraint solver, and we restrict the maximal value of the range, and the set of possible variables[4]. This provides us with an easily calculatable, finite set of constraints representing outcomes.

We randomly generated graphs by choosing a set of these outcomes, and forming a subsumption hierarchy from this set. Over the course of the simulation, we increased the size of the selected set, from 0% to 100% of the total set of constraints, incrementing by 20%, therefore increasing the number of nodes in the graph. For each percentage, we generated a fixed number of distinct graphs (in our simulations, we limited this to 100 graphs), and searched for every constraint in the constraint space using three different search algorithms: linear search, a depth-first search, and a depth-first search using variable propagation. For each search, we recorded the number of entailment operations that were performed, under the assumption that entailment is the most expensive operation in the search.

The results are summarised in Figure 3 with the 80% of the constraints being used. All searches appear to have $O(n)$ complexity as an average case, however, one can see improvements in the search time for both depth-first search implementations, especially for the depth-first search with variable propagation. Depth-first search without variable propagation performs approximately n entailments, and depth-first search with variable propagation performs approximately $\frac{1}{7}n$ entailments. Altering other parameters such as the amount of the constraint space used, and the set of variables in the constraint system gives different numbers here, but the general outline of the graph remains the same.

5.3 Discussion

From our analysis, we see that the overall complexity of insertion is increased for our method, while the average number of entailments in a search is decreased. A trade-off must be made between these two. Clearly, if the ratio of insertions to searches is high, then the average complexity is reduced. However, if this is the other way around, or the ratio is close to 1:1, then the average complexity is increased. It is our belief that searching would typically be performed considerably more often than insertion, so we believe that our method is superior to a linear search for anything other than the smallest of protocol libraries. Also, insertion of protocols into a library of protocols may be undertaken by dedicated agents not engaged in seeking protocols at run-time for interaction with other agents,

[4] We experimented with changing these two parameters, but with little change in the results.

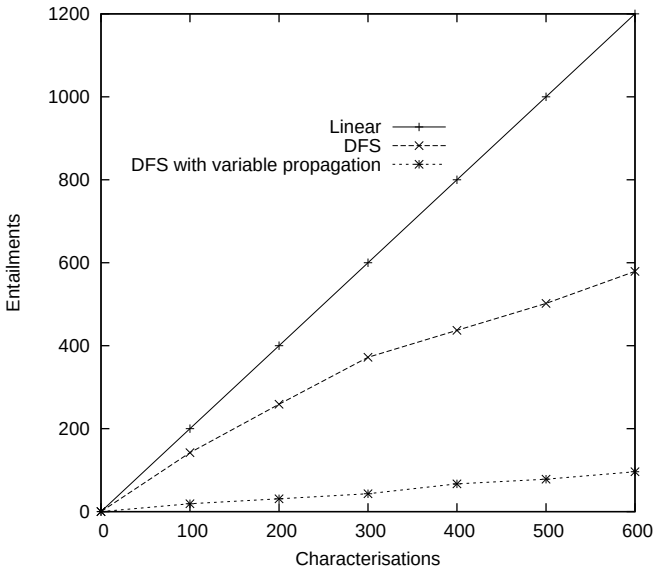

Fig. 3. Number of entailments for each search method.

and thus protocol-library insertion may be separated temporally, geographically and control-wise from protocol-library search.

Clearly, one of the questions that can arise is related to the formation of graphs. Given a set of characterisations, it is possible that none of the characterisations are related via the entailment operator, and we are therefore left with only source nodes. From a perspective of searching, this would be equivalent to a linear search. However, it is our belief that, given a set of characterisations, there will be enough relationships between characterisations to make our case worthwhile. After all, one of the visions of first-class protocols is that agents can choose different protocols that achieve the same goal, but which differ on other aspects.

Further improvements can be made to the above results by ordering the source nodes of the graph into a DAG themselves, using the propagated variables at the nodes as values, and the subset operator as the partial order. Rather than search through all of the source nodes, one can search the source nodes as a graph, only visiting the children if the source node's variables are a superset of the variables in the goal. If the current node is not applicable, and its children have *fewer* variables, then the children cannot be applicable either.

6 Related Work

There are many approaches to specification of first-class protocols, such as Yolum and Singh's commitment machines framework [12]. Space prevents us from a dis-

cussion of these; however, McGinnis and Miller [4] gives detailed presentation of the state-of-the-art in this area, and presents the advantages and disadvantages the different approaches, including $\mathcal{RASA}$. The general idea of annotation and matching could be applied to protocols specified in these languages, and indeed, could be used to structure and search plan libraries for agents.

As far as the authors are aware, there has been no investigation into the structuring and searching of either protocol or plan libraries for agents. Libraries, such as the plan libraries found in the Procedural Reasoning System [2], are searched sequentially, and offer no support for other techniques. In addition, the matching of plans to goals is usually testing whether the plan postcondition unifies with the goal, which is different to our approach of matching, due to us using constraint languages rather than logical languages, and using maximal postconditions.

Subsumption lattices have been used to reduce search space in inference [11] and in logic programming languages; for example, Muggleton [10] maintains a store as a subsumption lattice, and searches only the relevant parts of the lattice to match an inductive logic programming query. A subsumption hierarchy is similar to this, although it is not required to form a complete lattice. Logic programming languages take advantage of the lattice property and the existence of least upper bounds.

7 Conclusion

In this paper, we present a method for structuring and searching protocol libraries such that the search space for protocols is reduced. Using a partial order over the outcomes of protocols, libraries are sorted into directed graphs, and a depth-first search is used to find all protocols that achieve a given goal. Such an approach is more efficient than storing protocol libraries in a linear list, due to the depth-first search pruning of branches in the search process.

The average-case complexity of the depth-first search is $O(n)$, the same as a linear search; however, an experimental analysis of the two methods showed that the depth-first search performs fewer expensive entailment operations than are performed by a linear search. It is our assertion that these operations are the bottleneck of a search, and from this, we conclude that our structuring and searching method is superior to a linear search for anything other than the smallest of protocol libraries.

The work in this paper is another step towards achieving a vision of first-class protocols. However, before our full visions are realised, significant further work is required. In other work [9, 7], we have looked at issues such as verification of protocol specification, and of protocol compositions. In addition, meta-protocols are needed that allow agents to propose and negotiate which protocols are to be used, and suitable protocols for doing so will be investigated. To develop and test these ideas, we plan a prototype implementation in which agents negotiate the exchange of information using protocols specified using the $\mathcal{RASA}$ framework.

Acknowledgments

The authors are grateful for financial support from the EC through the PIPS (IST-FP6-507019) project and from the UK EPSRC through the *Market-Based Control of Complex Computational Systems* project (GR/T10657/01).

References

1. De Boer, F.S., Gabbrielli, M., Marchiori, E., Palamidessi, C.: Proving concurrent constraint programs correct. ACM Transactions on Programming Languages and Systems 19(5), 685–725 (September 1997)
2. Georgeff, M.P., Lansky, A.L.: Reactive reasoning and planning. In: Proceedings of the 6th National Conference on Artificial Intelligence. pp. 677–682 (1987)
3. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press, Cambridge, MA, USA (2000)
4. McGinnis, J., Miller, T.: Amongst first-class protocols. In: Artikis, A., O'Hare, G., Stathis, K., Vouros, G. (eds.) Engineering Societies in the Agents World VIII. LNAI, vol. 4995, pp. 208–223 (2007)
5. Miller, T., McBurney, P.: Using constraints and process algebra for specification of first-class agent interaction protocols. In: O'Hare, G., Ricci, A., O'Grady, M., Dikenelli, O. (eds.) Engineering Societies in the Agents World VII. LNAI, vol. 4457, pp. 245–264 (2007)
6. Miller, T., McBurney, P.: Annotation and matching first-class agent interaction protocols. In: Padgham, L., Parkes, D., Mueller, J.P., Parsons, S. (eds.) Proceedings of the Seventh International Conference on Autonomous Agents and Multi-Agent Systems. pp. 805–812. Estoril, Portugal (May 2008)
7. Miller, T., McBurney, P.: On illegal composition of first-class agent interaction protocols. In: Dobbie, G., Mans, B. (eds.) Proceedings of the Thirty-First Australasian Computer Science Conference. CRPIT, vol. 74, pp. 127–136. ACS (2008)
8. Miller, T., McBurney, P.: Characterising and matching iterative and recursive agent interaction protocols. In: Proceedings of the Ninth International Conference on Autonomous Agents and Multi-Agent Systems (2010), *(In Press)*
9. Miller, T., McBurney, P.: Propositional dynamic logic for reasoning about first-class agent interaction protocols. Computational Intelligence (2010), *(In Press)*
10. Muggleton, S.: Inverse entailment and progol. New generation computing 13(3), 245–286 (1995)
11. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall (2003)
12. Yolum, P., Singh, M.P.: Commitment machines. In: Meyer, J.J.C., Tambe, M. (eds.) Proceedings of the 8th International Workshop on Agent Theories, Architectures, and Languages. LNCS, vol. 2333, pp. 235–247. Springer (2002)