Distributed Problem-Solving as Concurrent Theorem Proving*

Michael Fisher[1] and Michael Wooldridge[2]

[1] Department of Computing, Manchester Metropolitan University, Manchester M1 5GD, United Kingdom
M.Fisher@doc.mmu.ac.uk
[2] Mitsubishi Electric Digital Library Group, 103 New Oxford St. London WC1A 1EB, United Kingdom
mjw@dlib.com

Abstract. Our principal aim in this paper is to describe how distributed problem solving may fruitfully be viewed as concurrent theorem proving. Not only does this approach provide an expressive representation technique for a range of distributed problem solving algorithms, but its powerful components allow the investigation of algorithms not yet captured within current systems.
We begin by introducing a novel *agent-based* approach to concurrent theorem proving which will form the basis for a range of distributed problem-solving applications, and then describe Concurrent METATEM, a multi-agent programming language whose model of computation is closely related to that used within the theorem proving approach. An extended case study is then presented, wherein we demonstrate how a multi-agent planning system can be implemented within our general framework and show how extensions and refinements of the planning system can easily be accommodated within our framework. Finally, we conclude with a detailed discussion of related work, from both the multi-agent systems community and the (concurrent) theorem proving community.

Topic areas: conceptual and theoretical foundations; multi-agent planning; cooperation.

1 Introduction

Problem solving is a fundamental issue in AI and, along with game-playing, is perhaps the oldest research topic in the discipline. The view of problem solving as *theorem proving* has a long and influential history in AI, going back at least to the work of Green [8]. This deductive view of problem solving has been particularly useful in AI planning research. *Distributed* problem solving (DPS), wherein a group of decentralized agents cooperate to solve problems, is perhaps the paradigm example of multi-agent activity, and is certainly the most-studied process in distributed AI. However, while many logic-based approaches to distributed AI have been described in the literature, we are aware

* This work was partially supported by EPSRC under grant GR/J48979.

of no work that has explicitly proposed viewing distributed problem solving as concurrent, agent-based theorem proving. This is perhaps due to the lack of an appropriate agent-based computational model for concurrent theorem proving.

The main aim of this paper is to propose that distributed problem solving may usefully be treated as concurrent theorem proving. To this end, we utilise a recently developed general framework for agent-based theorem proving, and demonstrate how this framework may easily be implemented in a multi-agent programming language. As an example of distributed problem solving, we consider distributed planning, representing both the basic system and various extensions and refinements within our framework.

The remainder of this paper is structured as follows. In §2, we present a general framework for concurrent theorem proving [7], and show how it can be used as the basis for distributed problem solving. In §3, we introduce Concurrent METATEM, a multi-agent programming language whose computational model is closely related to the agent-based theorem proving framework, making the language well-suited to implementing the technique. In §4, we present an extended case study, in which we show how a range of planning techniques may be represented in terms of the concurrent theorem proving technique introduced above. Finally, in §5 we discuss related work and provide concluding remarks.

2 A Framework for Concurrent Theorem Proving

The basic idea behind the approach is easily illustrated by means of a very simple example (from [7]). Consider the following set of propositional Horn clauses:

1. p
2. $\neg p \vee q \vee \neg r$
3. $\neg p \vee \neg q \vee \neg r$
4. $\neg p \vee r$

Using classical resolution, it is easy to derive the empty clause from this set. In our framework, theorem proving proceeds by allocating each clause $i \in \{1, \ldots, 4\}$ to an *agent*, Ag_i. The agents we consider are self contained reasoning systems, encapsulating both data and behaviour, and are able to execute independently of each other and communicate via broadcast message-passing [14]. In this theorem-proving context, these agents broadly behave as follows:

- any agent representing a clause containing just a positive literal should pass that information (via broadcast message-passing) to all other agents;
- upon receipt of a message, agents transform their own formulae on the basis of the information received, broadcasting any new literals generated, and reporting any contradiction produced.

Consider the simple example given above. As the agents begin executing, agent Ag_1, containing only the proposition p, broadcasts the message p to all other

agents. Once p has been received, each agent transforms its internal formula by applying classical (unit) resolution, and the configuration becomes:

(Ag_1) : p
(Ag_2) : $q \vee \neg r$
(Ag_3) : $\neg q \vee \neg r$
(Ag_4) : r

Agent Ag_4 then broadcasts r as new information. After this message reaches the other agents, they update their formulae, and the configuration becomes:

(Ag_1) : p
(Ag_2) : q
(Ag_3) : $\neg q$
(Ag_4) : r

Finally, agent Ag_2 then broadcasts q, and upon receipt of this message, Ag_3 generates a contradiction. Note that the theorem-proving activity is not dependent upon the order in which messages are sent. If the empty clause *can* be derived from the initial clauses, then it *will* be, as long as all messages sent are guaranteed to (eventually) arrive.

These agents have control over both their own execution, during which local deduction takes place, and their own message-passing behaviour. Since broadcast message-passing is used as the basic communication mechanism, other agents are able to view (and utilize) the intermediate deductions produced by each agent. Hence, global deductions are carried out *collectively* by the set of agents.

2.1 Generality and Correctness

Despite its simplicity, this approach is just as powerful as classical resolution: communication patterns in the concurrent system match the proof steps in a sequential resolution refutation. In the case of Horn clauses, the messages between agents correspond to positive literals while, in the case of full classical logic, the messages themselves correspond to Horn clauses. In addition, this approach can be extended to first-order logics. In [7], we show that the technique is in fact sound and refutation complete for classical logics; an easy extension to the basic technique above gives completeness for classical first-order logic.

In addition to basic details relating to local deduction, each agent may also contain information concerning behaviours not directly related to the core deduction process. In particular, the agents can represent and exchange information about both the heuristics currently being employed and the global organization of the agents. This additional behaviour is useful for added control, both in order to improve efficiency and to organize the agents in, for example cooperative, competetive or opportunistic structures [7].

2.2 Efficiency and Implementation

One potential criticism of the technique is the use of broadcast message passing, which is often regarded as too demanding of communication bandwidth to be used in practice. However, in spite of the use of broadcast, the system need not be flooded with messages. Not only is it possible to structure agents so that related information only occurs within one agent, but also, by grouping agents containing related parts of the problem-solving capability together, the number of messages generated can be greatly reduced [7]. Note that branching in the search space is here replaced by additional broadcast messages. Thus, in architectures where broadcast is prohibitively expensive, the technique may prove to be inefficient. But most contemporary architectures provide efficient multicast mechanisms (indeed, many distributed operating systems are based upon this mechanism: see, for example, [2]).

Our problem-solving framework is based on the approach described above. Thus, when a problem-solving agent has some definite positive information, it broadcasts this to all other agents. Similarly, once an agent receives information, it can transform its internal representation, possibly generating further communication.

3 Implementing the Framework

Having outlined the general model of concurrent theorem proving, we now describe the high-level programming language in which problem-solving applications will be represented. A Concurrent METATEM system [6] consists of a set of concurrently executing agents, which communicate through asynchronous broadcast message-passing. The internal computation mechanism for an agent is provided by the execution of *temporal logic* formulae [1]. We begin by giving a brief overview of temporal logic, followed by an outline of the execution mechanism for temporal formulae.

Temporal logic can be seen as classical logic extended with modal operators for representing temporal aspects of logical formulae. The temporal logic we use is based on a linear, discrete model of time. Thus, time is modeled as an infinite sequence of discrete states, with an identified starting point, called 'the beginning of time'. Classical formulae are used to represent constraints within individual states, while temporal formulae represent constraints *between* states. As formulae are interpreted at particular states in this sequence, operators which refer to both the past and future are required. The future-time temporal operators used in this paper are as follows: the *sometime in the future* operator — $\Diamond\varphi$ is true now if φ is true *sometime* in the future; and the *always in the future* operator — $\Box\varphi$ is true now if φ is true *always* in the future. Similarly, connectives are provided to enable formulae to refer to the *past*. The only past-time temporal operators needed for the examples in this paper are as follows: the *sometime in the past* operator — $\blacklozenge\,\varphi$ is true now if φ was true in the past; the *beginning of time* operator — **start** is only true at the beginning of time; and the

strong last-time operator — $\bigcirc\varphi$ is true if there was a last moment in time and, at that moment, φ was true[3].

Concurrent METATEM uses a set of 'rules', couched in temporal logic, to represent agent's intended behaviour. These rules are of the form:

'past and present formula' $\Rightarrow$ 'present or future formula'

Consider the following rules, forming a fragment of an example Concurrent METATEM program.

$$\begin{aligned} \textbf{start} &\Rightarrow \mathit{achieves}(a) \\ \bigcirc \mathit{goal}(X) &\Rightarrow \Diamond \mathit{planned}(X) \\ \bigcirc \mathit{topgoal}(Y) &\Rightarrow \mathit{subgoal}(Y) \vee \mathit{fact}(Y) \end{aligned}$$

Here 'X' and 'Y' represent universally quantified variables. Thus, we see that $\mathit{achieves}(a)$ is made true at the beginning of time and whenever $\mathit{goal}(X)$ is true in the last moment in time, a commitment to eventually make $\mathit{planned}(X)$ true is given. Similarly, whenever $\mathit{topgoal}(Y)$ is true in the last moment in time, then either $\mathit{subgoal}(Y)$ or $\mathit{fact}(Y)$ must be made true.

An agent's program rules are applied at every moment in time (i.e., at every step of the execution) and thus execution in a Concurrent METATEM agent can be distinguished from the logic programming approach in that refutation is not involved in the computation process and the model for the formula contained within the agent is constructed by following the temporal rules *forwards* in time. Once the agent has commenced execution, it continually follows a cycle of reading incoming messages, collecting together the rules that 'fire' (i.e., whose left-hand sides are satisfied by the current history), and executing one of the disjuncts represented by the conjunction of right-hand sides of 'fired' rules.

Each agent contains an *interface* describing both the messages that the agent will recognise and those it may send. For example, the interface

$$\mathit{top}(\mathit{goal}, \mathit{achieves})[\mathit{planned}, \mathit{subgoal}]:$$

defines *top* to be the name of an agent in which $\{\mathit{goal}, \mathit{achieves}\}$ is the set of messages the agent will accept, and $\{\mathit{planned}, \mathit{subgoal}\}$ defines the set of messages the agent can send.

For a more detailed description of the execution mechanism underlying Concurrent METATEM, see [1, 6].

4 A Case Study: Distributed Planning

In order to provide a concrete illustration of our approach, we consider a particular variety of problem-solving, namely planning, and show how this can be represented within our model. We begin with an overview of the AI planning

[3] A number of other operators are provided in Concurrent METATEM, though as they are not required for this paper, they will not be mentioned here; see [1, 6].

problem; our presentation is relatively standard, and is based on [9]. First, we assume a fixed set of actions $Ac = \{\alpha_1, \ldots, \alpha_n\}$, representing the effectoric capabilities of the agent for which we are developing a plan. A *descriptor* for an action $\alpha \in Ac$ is a triple $(P_\alpha, D_\alpha, A_\alpha)$, where[4]:

- $P_\alpha \subseteq \mathcal{L}_0$ is a set of sentences of first-order logic that characterize the *precondition* of α;
- $D_\alpha \subseteq \mathcal{L}_0$ is a set of sentences of first-order logic that characterize those facts made *false* by the performance of α (the *delete list*);
- $A_\alpha \subseteq \mathcal{L}_0$ is a set of sentences of first-order logic that characterize those facts made *true* by the performance of α (the *add list*).

A *planning problem* (over Ac) is then determined by a triple $\langle \Delta, O, \gamma \rangle$, where:

- $\Delta \subseteq \mathcal{L}_0$ is a set of sentences of first-order logic that characterize the *initial* state of the world;
- $O = \{(P_\alpha, D_\alpha, A_\alpha) \mid \alpha \in Ac\}$ is an indexed set of operator descriptors, one for each available action α; and
- $\gamma \subseteq \mathcal{L}_0$ is a set of sentences representing the *goal* to be achieved.

A *plan* π is a sequence of actions $\pi = (\alpha_1, \ldots, \alpha_n)$. With respect to a planning problem $\langle \Delta, O, \gamma \rangle$, a plan $\pi = (\alpha_1, \ldots, \alpha_n)$ determines a sequence of $n+1$ *world models* $\Delta_0, \Delta_1, \ldots, \Delta_n$ where:

$$\begin{aligned} \Delta_0 &= \Delta \quad \text{and} \\ \Delta_i &= (\Delta_{i-1} \setminus D_{\alpha_i}) \cup A_{\alpha_i} \quad \text{for } 1 \leq i \leq n. \end{aligned}$$

A (linear) plan $\pi = (\alpha_1, \ldots, \alpha_n)$ is said to be *acceptable* with respect to the problem $\langle \Delta, O, \gamma \rangle$ if, and only if, $\Delta_{i-1} \models P_{\alpha_i}$, for all $1 \leq i \leq n$ (i.e., if the precondition of every action is satisfied in the corresponding world model). A plan $\pi = (\alpha_1, \ldots, \alpha_n)$ is *correct* with respect to $\langle \Delta, O, \gamma \rangle$ if, and only if, it is acceptable and $\Delta_n \models \gamma$ (i.e., if the goal is achieved in the final world state generated by the plan). The planning problem can then be stated as follows: *Given a planning problem $\langle \Delta, O, \gamma \rangle$, find a correct plan for $\langle \Delta, O, \gamma \rangle$.*

We will now demonstrate how the planning problem can be solved using the general concurrent theorem proving paradigm we described in §2. More precisely, in §4.1 we show how a Concurrent METATEM system can be generated to solve a planning problem in a top-down (goal-driven) manner. We then prove correctness of the approach. In §4.2, we give an alternative method for deriving a Concurrent METATEM system, that will generate a solution to the planning problem in a data-driven (bottom-up) fashion, while in §4.3 we consider refinements of the two approaches.

[4] We assume a standard first-order logic $\mathcal{L}_0$ with logical consequence relation '$\models$'.

top-goal(γ)	γ is a top-level system goal
goal(γ)	γ is a sub-goal
achvs(Δ, π)	plan π achieves Δ
plan(γ, π)	plan π is a correct plan for γ

Table 1. Domain Predicates

4.1 Goal-Driven (Top-down) Planning

In this section, we demonstrate how, from a planning problem $\langle \Delta, O, \gamma \rangle$, a Concurrent METATEM system that will solve the problem in a top-down, goal-driven fashion can be systematically derived. We begin with a discussion of the various predicates that will be used, and an overview of the derived system structure.

We use just four domain predicates (see Table 1). The predicate *top-goal*(...) represents the fact that its argument (a set of $\mathcal{L}_0$ sentences) is the top-level goal of the system. The unary predicate *goal*(...) is used to represent sub-goals; its argument is also a set of $\mathcal{L}_0$ sentences. The *achvs*(...) predicate takes two arguments, the first of which is a set of $\mathcal{L}_0$ sentences, the second of which is a plan; *achvs*(Δ', π) represents the fact that plan π, when executed from the initial world state Δ, will achieve Δ'. Initially, we shall assume that plans are linear (see §4.3 for more complex plans), and represent them using a PROLOG-like list notation. Finally, the predicate *plan*(...) is used to communicate a plan to the originator of the top-level goal: *plan*(γ, π) means that plan π, if executed in the initial world, will achieve γ.

Given a planning problem $\langle \Delta, O, \gamma \rangle$, the basic generated system will contain $|\Delta| + |O| + 1$ agents: one for each element of Δ and O, and one 'top-level' agent. The top-level agent takes in a request for a plan to achieve γ, and sends out a message that creates a sub-goal γ. For each operator description $(P_\alpha, D_\alpha, A_\alpha) \in O$, an agent α is created, which encapsulates knowledge about the pre- and post-conditions of α. This knowledge is represented by the two rules ($TO1$) and ($TO2$). The first of these, ($TO1$), is essentially a rule for sub-goaling: it is fired when a message is received indicating that a goal has been created corresponding to the post-condition of α; in this case, the rule causes a new sub-goal to be created, corresponding to the pre-conditions of α. At some stage, a sub-goal created by this process will correspond to an initial state of the world (otherwise, the top-level goal γ is not achievable). This is where the third type of agent plays a part. For each sentence $\varphi_i \in \Delta$, an agent $init_i$ is created, containing a rule which represents the fact that if ever φ_i is a sub-goal, it can be achieved by the empty plan, '[]'. When such a rule fires, this information is propagated by sending the message *achvs*($\{\varphi_i\}$, []). These *base* agents can also combine initial conditions, sending out composite '*achvs*' messages. Within each α agent, there will be a single ($TO2$) rule, characterizing the effect of executing α; this rule will fire when a message *achvs*(Δ', π) is received, such that

Δ' matches the pre-condition of α. When fired, the rule will cause a message $achvs(\Delta'', [\alpha \mid \pi])$ to be broadcast, where Δ'' is the world-model obtained by executing α in Δ'.

We shall now describe the derived system (and in particular, the agents and rules used) in more detail.

Top-level agent: For $\langle \Delta, O, \gamma \rangle$, we create a top-level agent as follows.

$$
\begin{array}{ll}
top\text{-}level(top\text{-}goal, achieves)[plan, goal]: & \\
(TG1) & \bigcirc\!\!\!\!\bigcirc\, top\text{-}goal(\gamma) \Rightarrow goal(\gamma); \\
(TG2) & \bigcirc\!\!\!\!\bigcirc\, achvs(\Delta, \pi) \wedge \blacklozenge\, top\text{-}goal(\gamma) \wedge (\gamma \subseteq \Delta) \Rightarrow plan(\gamma, \pi).
\end{array}
$$

The agent *top-level* will accept a 'request' for a plan in the form of a message $top\text{-}goal(\gamma)$, where γ is the goal, as above. The rule $(TG1)$ then simply propagates γ as a sub-goal. The predicate *top-goal* would be given to the system by a user. Rule $(TG2)$ simply characterizes the *plan* predicate: π is a correct plan for γ if π achieves γ. When the *top-level* agent is informed of a plan π that achieves the top-level goal γ, it sends a message $plan(\gamma, \pi)$, indicating that a plan for the goal has been found. Thus, rule $(TG1)$ represents the *input* to the system, whereas $(TG2)$ represents the *output*.

Base agents: Given an initial world model $\Delta = \{\varphi_1, \ldots, \varphi_m\}$, we generate m agents, $init_1, \ldots, init_m$, each containing a rule $(TB1)$ showing that the initial conditions are achieved by the empty plan, together with a rule allowing the combination of relevant initial conditions $(TB2)$.

$$
\begin{array}{ll}
init_i(goal)[achvs]: & \\
(TB1) & goal(\varphi_i) \Rightarrow achvs(\{\varphi_i\}, [\,]); \\
(TB2) & \bigcirc\!\!\!\!\bigcirc\, achvs(\Delta', [\,]) \wedge (\varphi_i \notin \Delta') \Rightarrow achvs((\Delta' \cup \{\varphi_i\}), [\,]).
\end{array}
$$

Action agents: For each operator description $(P_\alpha, D_\alpha, A_\alpha) \in O$, where $A_\alpha = \{\varphi_1, \ldots, \varphi_m\}$ and $P_\alpha = \{\psi_1, \ldots, \psi_n\}$, we create an agent α as follows.

$$
\begin{array}{ll}
\alpha(goal, achvs)[goal, achvs]: & \\
(TO1) & goal(\varphi_1) \vee \cdots \vee goal(\varphi_m) \Rightarrow goal(\psi_1) \wedge \cdots \wedge goal(\psi_n); \\
(TO2) & \bigcirc\!\!\!\!\bigcirc\, achvs(\Delta', \pi) \wedge (P_\alpha \subseteq \Delta') \Rightarrow achvs(((\Delta' \setminus D_\alpha) \cup A_\alpha), [\alpha \mid \pi]).
\end{array}
$$

Rule $(TO1)$ generates sub-goals: if a sub-goal is received that matches against the post-condition of α, then this rule causes the pre-conditions of α to be propagated as sub-goals. Rule $(TO2)$ defines the effect that action α has upon an arbitrary state that satisfies its pre-condition. This rule effectively restricts us to linear plans — we consider non-linear planning in §4.3.

It is important to note that, while this approach may seem inefficient at first, *achvs* messages are only initiated for members of Δ that are required for one of the possible plans.

Correctness In this section, we prove that the approach to top-down planning discussed above is *correct*, in that: (i) any plan generated by the system is correct, and (ii) a system is guaranteed to eventually generate a plan for the top level goal γ. Alternaively, the correctness of this planning approach can be established via correspondence to the (complete) concurrent theorem-proving system [7].

Theorem 1. *Any plan generated by the system given above is correct. More precisely, if the message achvs(Δ', π) is broadcast in a system derived from a problem $\langle \Delta, O, \gamma \rangle$, then π is a correct plan for Δ'.*

Proof. By induction on the structure of plans. The base case is where π is empty; a message *achvs*$(\Delta', [])$ will only be sent by an $init_i$ agent, in which case Δ is true in the initial world, and will clearly be achieved by the empty plan. Next, suppose that π is of the form $[\alpha \mid \pi']$, and that if *achvs*(Δ'', π') is sent, then π' is correct for Δ''. If *achvs*$(\Delta', [\alpha \mid \pi'])$ is subsequently sent, then it must originate from a $(TO2)$ rule within the agent α. In this case, it is easy to see from inspection of $(TO2)$ that $[\alpha \mid \pi']$ is correct for Δ'.

Theorem 2. *If there exists a solution to the problem $\langle \Delta, O, \gamma \rangle$, then eventually, a system derived as above will find it. More precisely, if there exists a solution to $\langle \Delta, O, \gamma \rangle$, and the message top-goal(γ) is sent, then eventually a message achvs(γ, π) will be sent.*

Proof. By induction on the length of successful plans. The base case is that γ is directly achieved by the initial conditions of the system, and thus *goal*(γ) generates an appropriate *achvs*$(\Delta', [])$ message (where $\gamma \subseteq \Delta'$). Assuming that all problems requiring plans of length $n - 1$ can be solved, we assume that the plan $\alpha_1, \ldots, \alpha_n$ achieves the goal γ. Here, the message *goal*(γ) reaches agent α_n, which recognises γ and broadcasts appropriate subgoals. By the induction hypothesis, the subgoals will be solved and *achvs*(Δ'', π) will eventually be received by α_n (where $P_\alpha \subseteq \Delta''$). The α_n agent will then broadcast the solution to γ.

Theorems 1 and 2 together imply that a Concurrent METATEM program derived from problem $\langle \Delta, O, \gamma \rangle$ using the above scheme will be *totally correct*.

4.2 Data-Driven (Bottom-up) Planning

The operation of most implemented planners corresponds to the basic approach developed in the preceding section, in that they are goal-driven. Of course, there is an alternative, whereby a plan is developed in a data-driven manner. Many of the concepts are similar to the top-down planner (e.g., the various domain predicates retain their meaning). For this reason, our presentation will be somewhat more terse. Given a planning problem $\langle \Delta, O, \gamma \rangle$, we now generate a Concurrent METATEM system containing $|O|+2$ agents: one top-level agent, (as above), one agent for Δ, and one agent for each element of O. The system works by forward chaining from the initial state of the world, generating all possible

plans and their consequences. Eventually, the desired plan will be generated. However, given that there are $|O|!$ simple linear plans possible for operators O, it is not difficult to see that this form of plan generation will, in general, be impractical.

Top-level agent: This agent simply awaits a plan achieving the goal γ.

$$\begin{array}{ll} & \mathit{top\text{-}level}(\mathit{top\text{-}goal}, \mathit{achvs})[\mathit{plan}]: \\ (BG1) & \bigcirc \mathit{achvs}(\Delta', \pi) \wedge \blacklozenge\, \mathit{top\text{-}goal}(\gamma) \wedge (\gamma \subseteq \Delta') \Rightarrow \mathit{plan}(\gamma, \pi). \end{array}$$

Base agents: Given the initial world $\Delta = \{\varphi_1, \ldots, \varphi_m\}$, we now generate an agent, *init* which broadcasts all the relevant initial information[5]. Again, the agent contains a rule allowing the combination of relevant initial conditions ($BB2$).

$$\begin{array}{ll} & \mathit{init}()[\mathit{achvs}]: \\ (BB1) & \qquad\qquad\qquad\qquad \mathbf{start} \Rightarrow \bigwedge_{i=1}^{m} \mathit{achvs}(\{\varphi_i\}, [\,]); \\ (BB2) & \bigcirc \mathit{achvs}(\Delta', [\,]) \wedge (\varphi_i \notin \Delta') \Rightarrow \mathit{achvs}((\Delta' \cup \{\varphi_i\}), [\,]). \end{array}$$

Action agents: For each operation descriptor $(P_\alpha, D_\alpha, A_\alpha) \in O$ we generate an agent α, as follows:

$$\begin{array}{ll} & \alpha(\mathit{achvs})[\mathit{achvs}]: \\ (BO1) & \bigcirc \mathit{achvs}(\Delta', \pi) \wedge (P_\alpha \subseteq \Delta') \Rightarrow \mathit{achvs}(((\Delta' \setminus D_\alpha) \cup A_\alpha), [\alpha \mid \pi]). \end{array}$$

The rule ($BO1$) is identical to ($TO2$), above. Thus, there are no rules for decomposing a goal to produce sub-goals. Goals can only be solved by the required combination of plan elements being generated bottom-up.

Again, the correctness of this approach can be easily shown.

4.3 Refinements

We will now briefly outline a few possible refinements to the basic planning mechanisms discussed in the previous sub-sections.

Uni-directional top-down planning: When considering simple linear plans, generated using the top-down approach in §4.1, there is often no need to pass messages back through the action agents to achieve the final plan. Now, we extend the *goal* predicate with a second argument in which the partial plan for the current goal is stored. If the top-level agent broadcasts $\mathit{goal}(\gamma, [\,])$, then each action agent need only have the following rule for producing subgoals.

$$\mathit{goal}(\gamma, \pi) \;\wedge\; (A_\alpha \cap \gamma \neq \emptyset) \;\Rightarrow\; \mathit{goal}(\bigwedge_{i=1}^{n} \psi_i, [\alpha \mid \pi]).$$

[5] We here choose to use one base agent, rather than m base agents, in order to reinforce the fact that information can be distributed amongst agents in a variety of ways.

Thus, if any post-condition of the action occurs within a goal, then a new sub-goal corresponding to the pre-condition is generated and the current partial plan is extended with the action.

Each base agent now broadcasts the plan if it can reduce the goal completely:

$$goal(\varphi_i, \pi) \Rightarrow plan(\pi).$$

In this way, once a goal is completely decomposed, the second argument to *goal* must hold the plan that achieves the goal.

Non-linear planning: The top-down and bottom-up planners sketched above generate basic linear plans. An obvious extension is to develop *non-linear* plans. To provide this extension, we simply allow the preconditions of an action to be satisfied by states derived from different routes. Thus, assuming P_α is a set of n literals, $\varphi_1, \ldots, \varphi_n$, we simply change (BO1)/(TO1) to be

$$\left.\begin{array}{l} \bigcirc achvs(\Delta_1, \pi_1) \wedge (\varphi_1 \in \Delta_1) \quad \wedge \\ \quad \ldots \qquad \ldots \\ \bigcirc achvs(\Delta_n, \pi_n) \wedge (\varphi_n \in \Delta_n) \quad \wedge \\ \Delta = \Delta_1 \cup \cdots \cup \Delta_n \wedge consistent(\Delta) \end{array}\right\} \Rightarrow achvs(((\Delta \backslash D_\alpha) \cup A_\alpha), [\alpha \mid [\pi_1, \ldots, \pi_n]])$$

where *consistent* checks the satisfiability of a set of (usually ground) sentences.

Grouping and Efficiency: Until now, we have not considered the possibility of providing any additional structure within the agent space. Here, we briefly outline a mechanism, based upon *grouping* [11, 6], whereby detailed structure can be provided, limiting the extent of broadcast communication. The idea underlying the notion of an *agent group* is that each agent may be a member of several groups, and when an agent sends a message, that message is, by default, broadcast to all the members of its group(s), but to no other agents. Thus, if such groups are constructed appropriately, then the communication *between* groups will be more restrained than the communication *within* groups. In this case, it would be natural to implement such a scheme by limiting each group to one processor if possible.

Consider the simple top-down planning approach outlined in §4.1. Here, even if a plan never requires certain operators or initial conditions, broadcast messages will still be sent to these irrelevant items. If we are able to partition the agents space so that agents we are certain will not be needed in the final plan are excluded from a group, then broadcast communication may be effectively limited. The small the group produced, the less communication is required.

As a simple example of this, consider the set of clauses presented in §2. By grouping clauses 2 and 3 together into a sub-group which can receive messages, but not send them, we can ensure that messages relating to p and r are allowed through to Ag_2 and Ag_3, while q messages will never be allowed out of this sub-group. In this way, communication regarding q is localised and broadcast message-passing is reduced.

While grouping has many practical advantages, it can obviously lead to incompleteness. It is important that any heuristics used to group agents together are shown to retain the correctness of the problem-solving system. Thus, much of our current work, particularly with respect to concurrent theorem-proving, is centred around the development of appropriate heuristics.

Multi-agent aspects: While we have considered very simple agents whose rules are derived directly from the planning problem being solved, the model of distributed problem-solving we use is suitable for introducing more powerful control capabilities within the agents. Although it is initially counter-intuitive to add extra complexity to the problem-solving agents, this additional behaviour is often required not only to refine the specific behaviour of individual agents, but also to support organisation within the system as a whole.

While we will not consider the details of refined behaviour here, we note that techniques for representing and controlling agent-based systems in Concurrent METATEM have already been investigated [6]. In particular, agents can not only implement alternative internal problem-solving mechanisms, but can cooperate (for example sharing results and incorporating the benefits of cooperative search) or compete with other agents. With competition comes the ability to adapt, in a limited fashion, to changing circumstances. Combining both cooperation and competition, together with both agent grouping structures and the dynamic creation of new agents, we can begin to develop simple, yet evolving, agent societies capable of performing problem-solving [6].

5 Concluding Remarks and Related Work

We have introduced a model for distributed problem-solving, based upon an agent-based approach to concurrent theorem-proving. While space precludes both a longer discussion of the elements of the model and its application to other problem-solving domains, we believe the use of this model to represent varieties of distributed planning shows the potential for this approach. In addition to providing a consistent basis for distributed problem solving, this framework allows for the development of flexible and open agent-based systems, the use of broadcast communication being vital to the latter. While the utility of a logic-based approach for proving the correctness of distributed problem solving systems is clear in the case of planning, a wide range of further applications can be re-cast in this framework. In this way, the difficulties often encountered in establishing the correctness of dynamic distributed problem solvers may be alleviated.

Our future work in this area is to continue developing implementations and refinements of the approach (a prototype system already exists), to extend it to a wider range of distributed problem solving applications, and to show how such distributed systems can be formally derived from their single process counterparts. In addition, we are actively developing heuristics which can be used not only to group agents appropriately, but also to distribute information amongst agents.

Finally, we briefly consider related work in the section below.

5.1 Related Work

Our approach to concurrent theorem proving and distributed problem solving is somewhat similar to the blackboard model [5]. However, there are also significant differences: perhaps most importantly, our model allows for true concurrency, in that agents execute in parallel, and communicate via message-passing, rather than via globally accessible data structure. In terms of Smith's general classification of distributed problem solving, our framework is based on *result* sharing (as opposed to *task*-sharing) [13].

With respect to the underlying model of concurrent theorem-proving, while other systems share some features with our approach, the particularly close link between the operational model, communication and deduction, the possibility of dynamic creation, and the openness of the system makes it significantly different from previous systems.

In the DARES distributed reasoning system [10], agents cooperate to prove theorems. As in our model, information (set of clauses) is distributed amongst agents and local deduction occurs purely within an agent, but the agent can broadcast requests for further information. By contrast to our approach, not only is the number of agents static, but the opportunity for more sophisticated structuring of the agent space within the DARES system is absent. Further, the broadcast mechanism is not pervasive — it is only used to solicit new data when an agent stalls.

While the agents within the DARES system are all of the same type, one of the suggestions of that work was to consider different 'specialist' agents within the system. This form of system has been developed using the TEAMWORK approach, a general framework for the distribution of knowledge-based search [4]. While the number of agents within TEAMWORK is more fluid than in DARES, and more sophisticated structuring is provided through the concept of 'teams', the control within the system is centralised through the use of 'supervisor' agents. Also, in contrast to our model, less reliance is placed on broadcast communication.

The *clause diffusion* approach to concurrent theorem-proving [3] also partitions sets of clauses amongst agents. Unlike our framework, new clauses generated may be allocated to other agents. Thus, while new information generated in our approach is distributed by broadcast message-passing, this is achieved in clause diffusion via the migration of clauses. In contrast to our approach, clause diffusion is not primarily intended as a basis for the development of dynamic cooperative agent-based systems.

References

1. H. Barringer, M. Fisher, D. Gabbay, G. Gough, & R. Owens. METATEM: An Introduction. *Formal Aspects of Computing*, 7(5):533–549, 1995.

2. K. Birman. The Process Group Approach to Reliable Distributed Computing. TR91-1216, Dept. of Computer Science, Cornell University, 1991.
3. M. Bonacina & J. Hsiang. The Clause-Diffusion Methodology for Distributed Deduction. *Fundamenta Informaticae*, 24:177–207, 1995.
4. J. Denzinger. Knowledge-Based Distributed Search using Teamwork. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS)*, San Francisco, USA, 1995.
5. R. Englemore & T. Morgan (eds) *Blackboard Systems*. Addison-Wesley, 1988.
6. M. Fisher. A Survey of Concurrent METATEM — The Language and its Applications. In *First International Conference on Temporal Logic (ICTL)*, Bonn, Germany, 1994. (Published in *Lecture Notes in Computer Science*, vol. 827, Springer-Verlag).
7. M. Fisher. An Alternative Approach to Concurrent Theorem-Proving. In *Parallel Processing for Artificial Intelligence, 3*, Elsevier B.V., (in press).
8. C. Green. Application of Theorem Proving to Problem Solving. In *Proceedings of International Joint Conference on AI*, 1969.
9. V. Lifschitz. On the Semantics of STRIPS. In *Reasoning About Actions & Plans*, Morgan Kaufmann Publishers: San Mateo, CA, 1986.
10. D. MacIntosh, S. Conry, & R. Meyer. Distributed Automated Reasoning: Issues in Coordination, Cooperation, and Performance. *IEEE Transactions on Systems, Man and Cybernetics*, 21(6):1307–1316, 1991.
11. T. Maruichi, M. Ichikawa, and M. Tokoro. Modelling Autonomous Agents and their Groups. In Y. Demazeau and J. P. Muller, editors, *Decentralized AI 2 – Proceedings of the 2nd European Workshop on Modelling Autonomous Agents and Multi-Agent Worlds (MAAMAW '90)*. Elsevier/North Holland, 1991.
12. D. A. Plaisted and S. A. Greenbaum. A Structure-Preserving Clause Form Translation. *Journal of Symbolic Computation*, 2(3):293–304, September 1986.
13. R. Smith & R. Davis. Frameworks for Cooperation in Distributed Problem Solving. *IEEE Transactions on Systems, Man and Cybernetics*, 11(1):61–70, 1981.
14. M. Wooldridge & N. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

This article was processed using the LaTeX macro package with LLNCS style