MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo No. 673

December 1982
(Revised September 1986)

Cauldrons
An Abstraction for Concurrent Problem Solving
or
A Pandemonium for Representing Knowledge

Ken Haase

Abstract: The abstractions we have for serial programming are powerful; concepts like data types, variable binding, generalized operators, the "subroutine." We do not yet have the same sort of powerful abstractions for distributed computation, but I believe that the place to look for them is the same place that we found many of our abstractions for serial computation--- in our own minds. This research extends a tradition of distributed theories of *mind* into the implementation of a distributed problem solver. In this problem solver a number of ideas from Minsky's **Society of Mind** are implemented and are found to provide powerful abstractions for the programming of distributed systems. These abstractions are the ***cauldron***, a mechanism for instantiating reasoning contexts, the ***frame***, a way of modularly describing those contexts, and the ***goal-node***, a mechanism for bringing a particular context to bear on a specific task. The implementation of both these abstractions and the distributed problem solver in which they run is described, accompanied by examples of their application to various domains.

This report describes research done at both the MIT Artificial Intelligence Laboratory and the ATARI Systems Research Division. Support for MIT's artificial intelligence research is provided in part by the Office of Naval Research under Office of Naval Research contract N00014-79-C-0260

CONTENTS

FIGURES

1. Introduction

The abstractions we have for serial programming are powerful; concepts like data types, variable binding, generalized operators, the "subroutine." These concepts are our most powerful tools in the development and understanding of large systems for conventional serial machines. This paper presents a set of similar abstractions for parallel computation and describes a distributed problem solving language which implements them.

These abstractions are the ***cauldron***, a mechanism for organizing inference into distinct reasoning contexts; the ***frame***, a way of modularly describing the components of these contexts; and the ***goal-node***, a mechanism for bringing a particular reasoning context to bear on a specific task. The development of these abstractions, in addition to providing a working base for experiments in parallelism, gives a new perspective on the role of "representation" in reasoning systems.

1.1 Main Points

This paper develops three mechanisms for organizing large distributed reasoning systems:

> ***Cauldrons*** -- A chunk of computational activity containing a set of assertions and inferential mechanisms for manipulating them.
>
> ***Frames*** -- A way of grouping assertions into "chunks" of knowledge. For instance, a ***frame*** for a particular block contains knowledge -- in the form of propositional descriptions -- about that block.
>
> ***Goal-nodes*** -- A mechanism for invoking a collection of ***frames*** into a particular ***cauldron*** to perform some task.

A *cauldron* is a restricted problem solving context. In a cauldron, reasoning takes place separated from the main stream of computation, which ideally consists only of a system of interacting cauldrons.

The contents of a cauldron are assertions, some of which may be *computationally active* in the sense that they may produce side effects in response to changes or events in the cauldron. These side effects might be new assertions, remarks to the console, or the creation of other cauldrons. Pattern-invoked rules or inter-cauldron communcation channels are instances of these active assertions. The cauldron metaphor is meant to invoke the vision of an actively changing "brew" of rules and assertions interacting and combining to form new conclusions and results.

A *frame* is a collection of assertions which may be added to a cauldron. Since these assertions may be rules or other computationally active forms, the knowledge a frame contains may be either procedural or declarative. **The presence of this active knowledge allows a piece of knowledge to *contain its own interpretation*.** For instance, the statement "Jack is the brother of Jill" might share a frame with the rules of interpretation defining *what it means to be a brother.*

A frames' contents may be defined by pointers to other frames as well as by its explicit contents, allowing a frame to indirectly *include* other frames. The frame describing Jack -- for instance -- might indirectly include the rules and assertions defining "what it means to be a brother," which would interpret the assertions about brotherhood in Jack's description. In an identical manner, relations such as AKO can be interpreted in a manner *specific to the reasoning context.* Note that this inclusion mechanism makes no epistemological assumptions, but only describes contexts in which an epistemology may be defined and interpreted.

A *goal-node* is a way of tying frames and cauldrons together to solve a problem.[1] In a response to an explicit goal of the problem solver, a goal-node creates a *cauldron* to work on that goal and adds the assertional contents of a certain set of *frames* to that newly created cauldron.

For instance, when trying to place one block atop another, a triggered goal node will create a cauldron which contains frames for the blocks involved as well as a frame -- or frames -- describing the technique for making one block support another. *Since the assertions in a frame may be rules, a frame may contain the declarative procedure for peforming some task.*

These three mechanisms provide a facility for abstraction which declarative programming languages, even distributed ones, do not generally provide. The goal node makes possible the creation of reasoning subcontexts in much the same way that the subroutine allows creation of a variable-binding context for the execution of a procedure. In an extended analogy to traditional subroutine invocation, the cauldron takes the place of the stack frame, the frames take the place of binding environments *and* the function definition, and the goal-node takes the place of the calling mechanism.

1. "Solving a problem" is intended in its most general sense. It may refer to writing a sonnet, figuring out what a tool does, or solving an electrical network.

1.2 An Example

This example portrays a cauldrons-based reasoning program peforming tasks in the blocks world. The environment in which it operates is a classical blocks world where the program performs simple blocks world tasks, using frames and goal nodes to carry them out in a reasonably sophisticated manner.

Several important points are illustrated in this example:

o The cauldron invocation mechanism as a way of capturing the application of a particular technique to a given problem.

o The functionality of separating intentions into *needs* which are reasoned about and *goals* which are acted upon.

o The frame mechanism as a way of modularizing knowledge into *contours of relavance* based on what knowledge is useful for particular tasks.

o The frame-sharing mechanism as a way to communicate *relevant* information between cauldrons.

o The homogenous representation of both program and data as active or inactive elements of *frames*, allowing the simple attachement of censors and critics to arbitrary pieces of knowledge or procedure.

These points encompass both the mechanisms which this paper presents -- cauldrons and frames and goal-nodes -- and various "stylistic" principles which make programs using these mechanisms easily modifiable and extensible.

The blocks world starts out as in Figure 1. Initially, there is a single cauldron, BLOCKS-CAULDRON, to which tasks are given. Figure 2 presents this cauldron with its set of initial assertions, rules, frames, etc;

The first task given to the program is the straightforward problem of placing BLOCK1 on top of BLOCK2. A goal-node in BLOCKS-CAULDRON, triggered by the assertion of this goal, creates a subcauldron for fulfilling it. This subcauldron contains a reasoning context, made up of rules and assertions, tailored to the task at hand.

In this particular case, the cauldron contains the following elements:

Fig. 1. The Blocks World

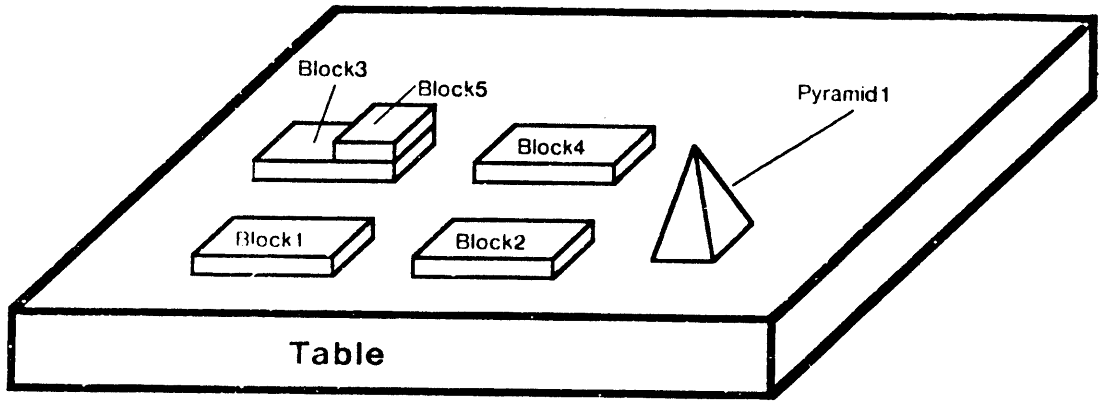
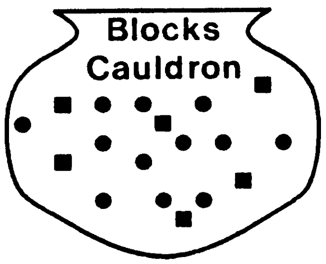


Fig. 2. Blocks Cauldron



o The physical details of BLOCK1 and BLOCK2. Their size, shape, color, position, and relations with adjoining blocks.

o What it "means" to be a block. This incorporates such axioms as "if a block doesn't support anything, it is clear", or that "if you are looking for space for a block (through an automated oracle connected to the blocks world simulator) you will generally find it".

o Techniques for interpreting new "sensory" knowledge. This is essentially a set of rules for transforming between the assertions of the blocks world simulator and the internal representations of the program.

o Basic techniques for the domain. This is knowledge which is shared by all techniques which BLOCKS-CAULDRON might invoke; it includes knowledge about how to know if you're finished with a task, principles for maintaining a consistent world model between separated cauldrons, and interlocks for unique resources (such as hands).

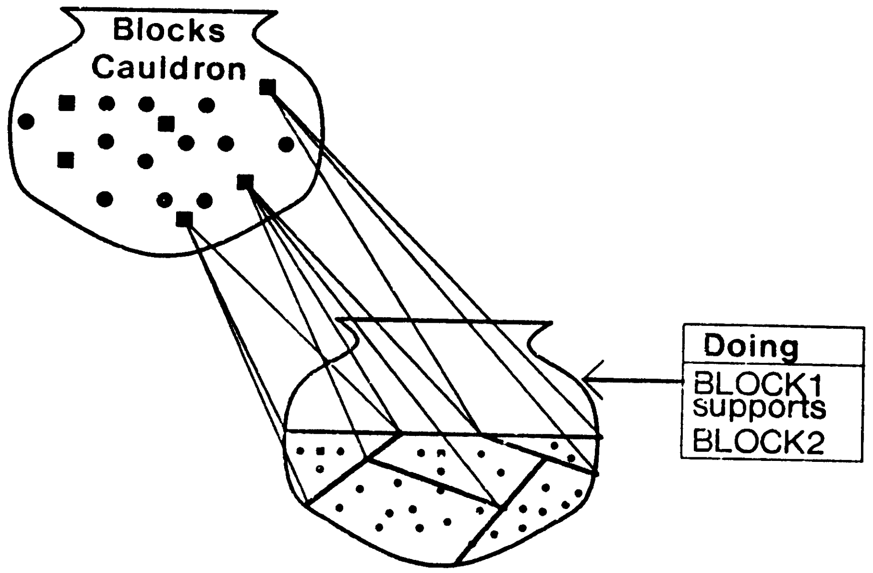
o Techniques for performing the task. This is a traditional blocks-world program which ensures preconditions, moves, grabs, moves, and lets go. These techniques are implemented as state-to-state rule-chains which fire off of results and preconditions to generate actions and new results. The handling of unsatisfied preconditions is demonstrated later in this example.

o Censors which are particular to the sort of task being performed; This is where knowledge such as the unsuitability of pyramids as supports is stored. In the current implementation the knowledge here is anecdotal, and does not take the form of complex "theories of support suitablity".

These individual "pieces of knowledge," stored as *frame assertions* in BLOCKS-CAULDRON, are "activated" into the newly created subcauldron. This process is depicted in Figure 3. *Activation* involves running the deduction mechanism over the new set of assertions and rules, with reference to those rules already in the cauldron.

The subcauldron now contains several sorts of "how-to" knowledge: procedural knowledge in the form of rule-chains, censor-knowledge in the form of simple condition-triggered rules, and prerequisite checkers

Fig. 3. Activation of a cauldron



for enabling the procedural rule-chains. To begin, the prerequisite checkers, noting that BLOCK2 can be grasped and that there is adequate space for it on BLOCK1, enable the support-making rule chain, which proceeds to pick up BLOCK2 and carry it over to BLOCK1, where it is released. The subcauldron, its goal acheived, now updates the appropriate frames in BLOCKS-CAULDRON (to reflect the changed state of BLOCK1 and BLOCK2, particularly) and dissolves itself.

Next, the program is given the more complex task of placing BLOCK2 on top of BLOCK3; this task is more involved because BLOCK5 is already on top of BLOCK3, as in Figure 4, and the program must recognize and remove this obstruction. When the goal of making BLOCK3 support BLOCK2 is recognized, a goal-node creates a cauldron, just as before, to pursue the goal. The contents of this cauldron, as before, is determined by the frames activated into it. This arrangement is depicted in Figure 5. The cauldron -- or to speak precisely, its interpreted contents -- first checks to see if it is okay to grasp BLOCK2, and seeing that it is, checks to see that there is space for BLOCK2 on BLOCK3, thus noting the presence of BLOCK5. Seeing the obstacle, the cauldron generates upwards to BLOCKS-CAULDRON a *need* for BLOCK3 to not support BLOCK5, including an explanation of why it is needed.

BLOCKS-CAULDRON, receiving notice of this *need*, and noting no evidence to suggest its improperiety, generates the *goal* of satisfying it. This new goal starts up another cauldron, as shown in Figure 6,

with an entirely different set of expertise from the first cauldron. *The contents of this* NOT-SUPPORTS

Fig. 4. The Blocks World after (MAKE (BLOCK1 SUPPORTS BLOCK2))

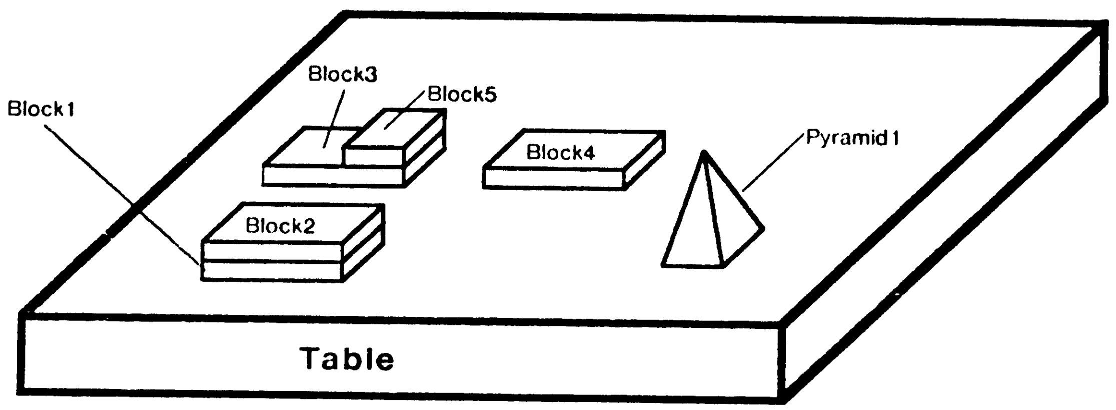


Fig. 5. A cauldron is created to make BLOCK3 support BLOCK2

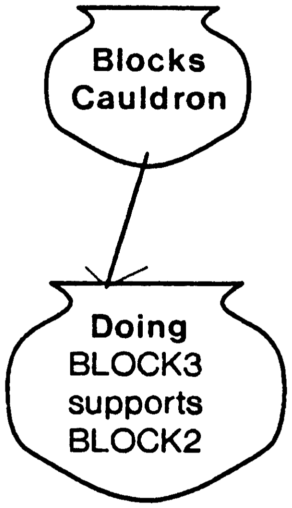
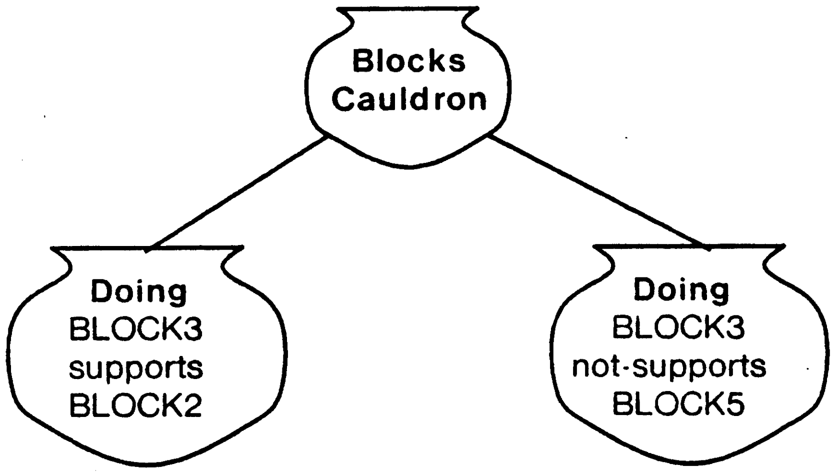
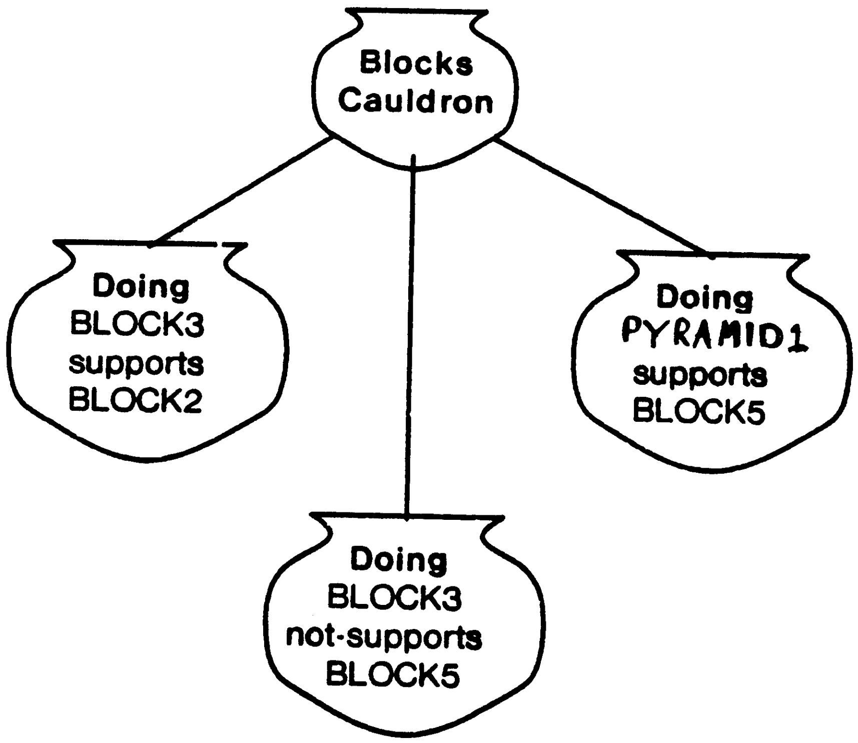


Fig. 6. Figuring out how to satisfy a prerequisite


Blocks Cauldron
Doing BLOCK3 supports BLOCK2
Doing BLOCK3 not-supports BLOCK5

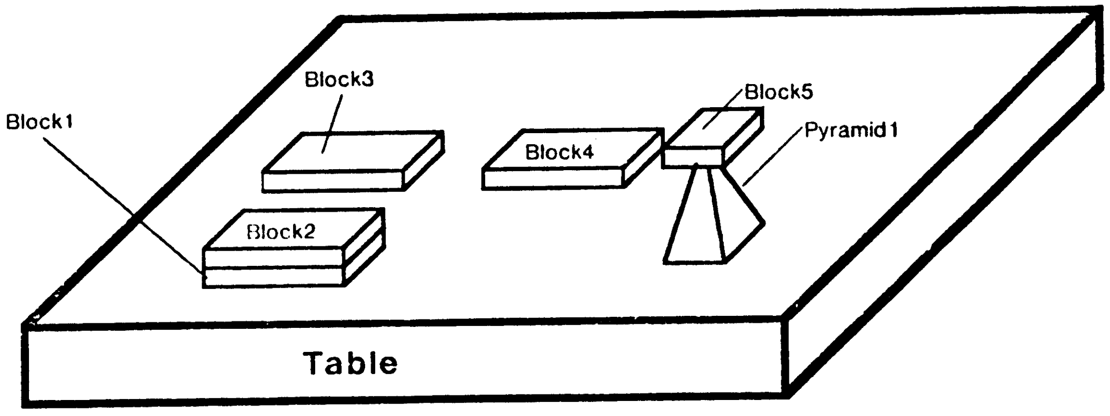
cauldron -- the rules and assertions which provide its expertise and expectations -- are very different from the contents of any MAKE-SUPPORTS *cauldrons.* The procedure represented in the NOT-SUPPORTS cauldron looks at all of the blocks and tries to find a safe place to discard BLOCK5 which wil not disrupt any ongoing tasks.[2] The NOT-SUPPORTS cauldron decides that BLOCK5 can be put on top of PYRAMID1, and generates a *need* for that support relation to exist. This is transformed into a goal which triggers the activation of still another cauldron using the same general MAKE-SUPPORTS knowledge as the first. This new subcauldron proceeds to pick up BLOCK5, carry it over to PYRAMID1, and attempt to release it. This arrangement of cauldrons is shown in Figure 7; Figure 8 depicts the blocks world at this moment. Unfortunately, BLOCK5 cannot rest securely on top of PYRAMID1 and this problem is reported by the blocks world simulator a statement about the instability of BLOCK5 asserted into the MAKE-SUPPORTS cauldron which attempted to

Fig. 7. Satisfying the prerequisite


Blocks Cauldron
Doing BLOCK3 supports BLOCK2
Doing PYRAMID1 supports BLOCK5
Doing BLOCK3 not-supports BLOCK5

2. Inter-task interference is mediated by a *protection* mechanism (as in HACKER [Sussman76]). When a cauldron begins to manipulate a block, it adds the statement that the block is protected to the frame for the block as well as to a repository BLOCK-PROTECTIONS frame in BLOCKS-CAULDRON. The NOT-SUPPORTS cauldron has this BLOCK-PROTECTIONS frame activated into it so that it can identify conflicts with tasks currently in progress.

Fig. 8. Trying to put BLOCK5 on top of PYRAMID1



release BLOCK5 on top of the pyramid. This assertion is recognized by the cauldron (this is the second MAKE-SUPPORTS cauldron) and a report of the difficulty is percolated up to BLOCKS-CAULDRON, with two results:

> o A *critic* is created to catch subsequent cauldrons about to attempt the same ill-founded goal (the goal of putting BLOCK5 on top of PYRAMID1). This critic is stored in the frames for BLOCK5 and PYRAMID1, which are both defined inside of BLOCKS-CAULDRON.
>
> o An assertion describing the error is added to the cauldrons which brought about the error-producing need (the NOT-SUPPORTS cauldron and the initial MAKE-SUPPORTS cauldron) in the first place.

Reacting to the error assertion, the NOT-SUPPORTS cauldron looks for an alternate location for BLOCK5, and decides to place it atop BLOCK4. This decision generates two assertions in BLOCKS-CAULDRON: an appropriate *need* for BLOCK4 to support BLOCK5 and a statement that the error of the misguided MAKE-SUPPORTS cauldron has been "handled". The second assertion leads to the dissolution of the erring MAKE-SUPPORTS cauldron, while the first generates a goal of having BLOCK4 support BLOCK5, which triggers the creation of a new cauldron working on that goal, as show in Figure 9.

This new cauldron, seeing that BLOCK5 is already grasped, moves to BLOCK4 and drops BLOCK5 on top of it. The blocks world is now as depicted in Figure 10. The cauldron responsible for this, its task done and having updated the appropriate frames in BLOCKS-CAULDRON, dissolves itself. Similarly, the

Fig. 9. Recovering from the error

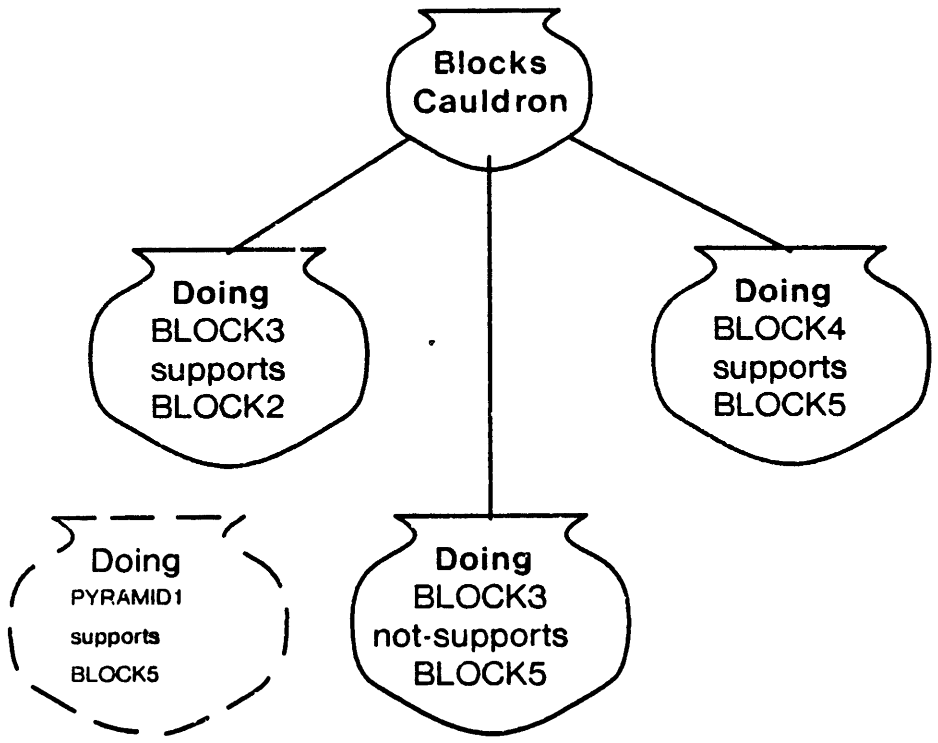
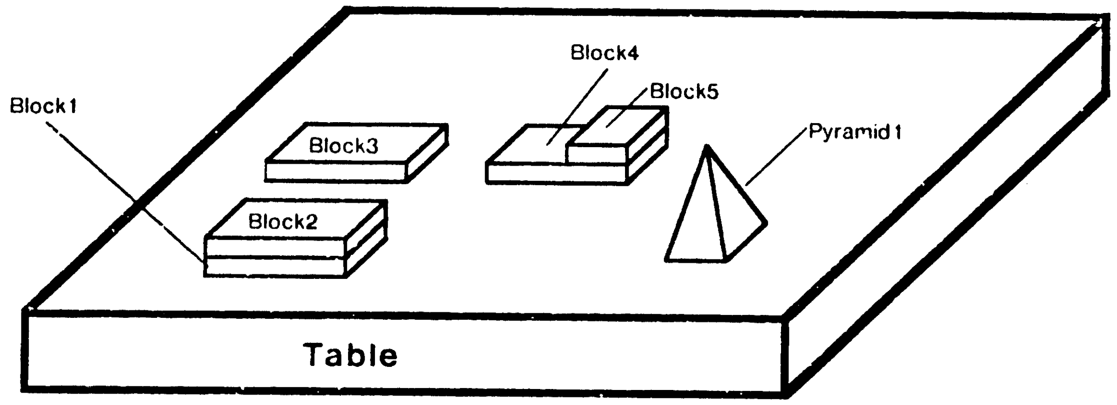

Blocks Cauldron
Doing BLOCK3 supports BLOCK2
Doing BLOCK4 supports BLOCK5
Doing PYRAMID1 supports BLOCK5
Doing BLOCK3 not-supports BLOCK5

Fig. 10. The blocks world after BLOCK3 has been cleared

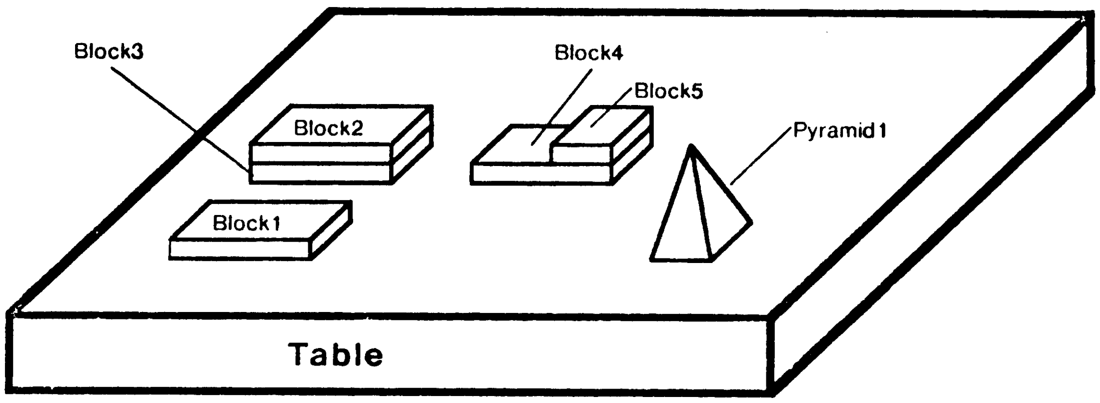


NOT-SUPPORTS cauldron, seeing its goal safely acheived, dissolves itself along with the others.

When a frame is changed, the cauldrons into which that frame is invoked are updated. For instance, when the frame for BLOCK3 is suitably changed by a cauldron affecting BLOCK3, the change is asserted into any cauldron into which the frame for BLOCK3 has been invoked. Thus, when the second MAKE-SUPPORTS cauldron (which tried to put BLOCK5 on top of PYRAMID1) picked up BLOCK5 from BLOCK3, it updated the frame for BLOCK3 to reflect the removal. When this fact reached the first MAKE-SUPPORTS cauldron (because it contained the frame for BLOCK3), the prerequisite checkers recognized that there was now space for BLOCK2 on BLOCK3, making it possible to move BLOCK2 as soon as the hand became free. Thus, when BLOCK5 is finally released, and the program's hand is free, the program moves over to BLOCK2, grabs it, carries it over to BLOCK3 and releases it. This done, the first MAKE-SUPPORTS cauldron dissolves itself after updating the appropriate frames in BLOCKS-CAULDRON, leaving the blocks world as in Figure 11.

Next we ask the program to place BLOCK5 on top of PYRAMID1. The program refuses, explaining (given the critic which was created earlier) that BLOCK5 would be unstable if this were attempted. If this goal had been the result of one of its own deliberations, for instance a NOT-SUPPORTS attempt, it would be able to recognized the expected error and try another approach or strategy. (For instance trying to find another space for the block).

Fig. 11. The Blocks World after (MAKE (BLOCK3 SUPPORTS BLOCK2))



1.3 Learning with Frames

This example has shown goal-nodes, frames, and cauldrons in action performing a series of tasks. The notions of frames and goal-nodes, however, support several powerful mechanisms for learning from experience and mistakes. This section offers a few notes (without extensive examples) on these mechanisms.

The first, implemented in an earlier version of the blocks world program presented here, recognized 'error assertions' like that produced by the misguided cauldron in the example. It then initiated a sub-cauldron to hypothesize a reason for the failure; this reason then became a *censor* applied to MAKE-SUPPORT activities in general. Unfortunately, the hypothesis generator, as first coded, tended to over-generalize and --- since the implementation contains no provisions for automatic truth maintenance --- eventually paralyzed the problem solver by the image of disaster at every turn.

The second, used more extensively in reasoning about connectivity in circuits, used the idea of *storing activation patterns* as a way of constructing new frames. A successful cauldron --- its goal acheived --- could collect its contents (the results --- in terms of intermediate conclusions and primed-to-fire rules) and make them (or add them to) frames in the cauldron above it. In particular, the circuit reasoning program computed a lot of fanouts in the process of trying to find breaks in a represented electric circuit. By adding these fanouts to the frame describing the ciruit, later analysis tasks could proceed quite quickly --- skipping endless chains of fanout specifications. In the blocks world, where the frames for blocks change so often, this was less effective; restoring past state could seriously confuse the problem solver rather than clarifying or improving its progress.

A later implementation of cauldrons might make more use of these facilities, with powerful tools for storing, pruning, and restoring partial states of the problem solver.

2. Restricting and Distributing Reasoning

This section describes the cauldron mechanism as a mechanism for restricting and distributing reasoning over several distinct inferential "processes". While not delving into the internals of an implementation, it details the motivations for its design and contrasts it with other distributed reasoning systems.

2.1 Cauldrons

This research develops a mechanism for defining and maintaining restricted problem solving computations called *cauldrons.* Cauldrons are explicitly defined problem solving contexts in a reasoning program. An individual cauldron consists of a declarative program interpreter roughly similar to AMORD [deKleer78], a database of assertions, rules and other structures on which this interpreter acts, and control information which it uses in interaction with other cauldrons.

The cauldrons' "contents" are the set of assertions on which the interpreter operates. Rules and special constructs such as side effect generators and inter-cauldron communication channels are simply special cases of assertions. These rules and active assertions may be examined and manipulated by other rules and active assertions in the cauldron just as easily as they manipulate the assertions and inerences of some particular domain.[3]

One fundamental assumption of this research has been the *active* role of knowledge in the reasoning process. The concept of apple has far more "attached" to it than merely color, shape, size, or flavor. It carries the knowledge that you perhaps shouldn't eat it when green, that you can cut out its soft spots with a knife, and that planting its seeds can make you feel exceptionally nature-loving. The view of knowledge as merely a collection of statements to be examined is impoverished because it excludes a vast array of censor-knowledge, procedure-knowledge and "contextual interpretation" knowledge. Casting knowledge as a passive "accessed" element of the reasoning process sterilizes a representation, draining it of expressibility. This is a fate which this research has tried to avoid by discarding the idea of "database access" as a distinct operation. Cauldrons

3. Kornfeld's Ether [Kornfeld82] lacks this feature; Rules, as sprites, are not normally looked at by other rules or sprites-- there is a clear separation between assertions and rules in Ether, something which cripples attempts at reflection or self modification. If a rule must determine what other rule led to some action or conclusion -- in order to either disable it for similar situations or generalize it to other situations -- the ability to examine rules with other rules is neccessary.

are not a database from which "facts" are fetched, they are a melting pot of techniques and knowledge into which "facts"[4] are *activated.* Rules, effectors, and examiners hence become only especially "active" assertions.

A given cauldron is able to manipulate other cauldrons through these "active" assertions. Through this mechanism, a cauldron can either create new cauldrons or make assertions in existing cauldrons. This ability of cauldrons to create other cauldrons defines a hierarchy of cauldron invocation. The arcs of this hierarchy describe a *parent-child* relation, where the cauldron created is referred to as the *child* of its creator *parent* cauldron. This hierarchy is intimately tied to the frame and goal-node abstraction presented in this paper. Generally the goal-nodes and frames which contribute to a given cauldron are present -- as single `FRAME` or `GOAL-NODE` assertions -- in their *parent* cauldron. Conversely of course, the frames and goal-nodes existing as assertions in a cauldron are generally used to instantiate and define its *children.*

While the primitive mechanisms for cauldron creation and activation are available to the rules and assertions of each cauldron, programs generally use the goal-node and frame abstractions sketched in the introduction.[5] The goal-node/frame mechanism supports a useful abstraction over the bare cauldrons implementation, providing a protocol for organizing knowledge and method.

2.2 Distinctions from other Distributed Reasoning Systems

This section will try to motivate some of the features described in the previous section by comparision with other schemes for distributed reasoning.

The Conniver [McDermott74] ctxt mechanism may be considered a subset of cauldrons, as it creates new subdomains by "layering" new assertions onto the database so that they can be "pushed" -- defining a new layer of context -- and "popped" -- returning to a previous layer. But the use of layered contexts, while gaining the "computational instantiation" aspect of cauldrons, fails as a restriction mechanism. Its problem lies in dragging the entire ancestor context into the computational arena.[6] This is cumbersome in most cases,

4. Where a fact may as easily be "how to paint a block" as "the block is red".
5. The goal-node and frame mechanisms were initially implemented using these explicit cauldron-manipulation primitives, and only after those explicit implementations had evolved were they wired into the implementation.
6. You can get around this problem by adding things to an empty context, but this usurps the hierarchy defined by context-creation, which is still important from a control standpoint.

but may sometimes be necessary-- a cauldron instantiation does have the option of copying the entire present cauldron into the new one. Conniver's layered context mechanism may be viewed as creating a *hierarchy of assumption* with new context layers being new branches from a single tree of deduction, while the cauldron mechanism defines a *hierarchy of invocation.*[7] The hierarchy of invocation reflects both representational abstraction and meta-knowledge between levels; ***a cauldron is "superior" to another if it interprets and uses its results.*** An inferior cauldron will usually know *more* about certain things than the superior cauldron which invoked it (such as how to perform its task), but will in turn know *less* about other things (such as why it was invoked in the first place). The cauldron hierarchy is determined not by accumulated layers of conclusions, but by invocation, with each invocation guided by knowledge about the purpose and performance of levels below it.

This design inherently reflects a philosophy for the control of parallel reasoning processes. While it is possible to implement a layered context mechanism in a cauldron system, the general philosophy is that usually this is the wrong thing to do. The design of cauldrons stresses a hierarchy of invocation control over a hierarchy of accumulated assumptions.

TMS systems such as Doyle's [Doyle78] may be viewed as a systematic dissolution of the Conniver tree of cumulative assertions. Using a TMS both gets around the costs of instantiating new contexts for each push and, more importantly, permits the removal of an individual "pushed" assertions in a long chain of "pushes" without having to regenerate the entire chain. It does this by keeping track of the "local causes" of each push and further by making each assertion be a push. Thus the tree of pushes is broken into a collection of the local interactions (dependencies) for each push. The problem with this dissolution is that the Conniver hierarchy of assumption also contains control information and results of considerations which the TMS may well remove. The solution to this in AMORD[deKleer78] is to explicitly define certain types of assertions to be sancrosanct as far as the TMS is concerned. I argue that it is preferable to retain an explicit hierarchy of *control* -- storing control information and results at different levels in the cauldron invocation hierarchy -- while performing maintenance of hypotheticals throught explicitly accessible mechanisms on each level (within each individual cauldron).

7. Cauldrons actually supports a **heterarchy** of invocation, allowing a cauldron to have more than one parent. While I can easily imagine schemes in which this would be useful, I have yet to actually use such a feature in any slightly real domain. Such a "godparent" relation might be useful when a number of separate cauldrons are interested in the results of a given computation; each interested cauldron can be a "godparent" to a single cauldron performing the computation.

The **ACTORS** paradigm [Hewitt76] differs from cauldrons in two distinct ways: the size of the agents involved and the existence of a clear control structure between agents at different levels and tasks.

I imagine a cauldron as far larger than an actor in terms of both memory and computational activity. In general, I envision a cauldron to contain on the order of one hundred rules and perhaps three to six times that many assertions which are not rules. ("Rule" here refers to any sort of "active" assertion.) This number -- an off the cuff calculation based on when the current implementation becomes unwieldy in terms of efficiency or accessibility to debugging -- refers to each cauldron's *ultimate* size, rather than to the size of the kernel which creates it.

Computation on a cauldron-level resembles the actor model, with each cauldron as a message passing entity, but most of the real computation in a cauldrons based reasoner goes on inside of the individual cauldrons, in an environment of rules and assertions. This distinction catches on a commonly recognized flaw in the **scientific community metaphor**, a recent explication of the actor model. The scientific community does not produce theories, *scientists do.* Thus, if the "actors" are scientists, each scientist must have enough computational power (in terms of knowledge as well as cycles) to generate a theory. The theory does not necessarily have to be *good*, but it must have the internal structure, in terms of dependencies and motivations, of a complete theory. The difficulty is that on the computational side of the scientific community metaphor, there is no chunk of computational mechanism sufficiently large to serve as a scientist.[8] I view the cauldron as a "chunk of mechanism" large enough to both develop conclusions and to *motivate* those conclusions. The typical ***actor*** is not sufficiently complex to satisfy this requirement.

For instance, in the blocks world example of Section 1.3, a subcauldron issues a *need* for some relation to exist, and explains that need. If a cauldron needs to choose between the multiple needs of its children, this explanatory justification is necessary to making correct judgements. But the subcauldrons can only provide this information if they are complex enough to realize that:

Prerequisite = => Action becomes possible = => Goal may be achieved

The typical actor or sprite or rule is much too simple and small to possess such complex knowledge. This

8. The flaws described here are not flaws with the actor model, but with the characterization of a community of simple actors as a scientific community. The actor mechanism -- like the lambda calculus -- provides a battery of powerful terms and concepts for describing computation. For many purposes, it is useful to describe the rules in a given cauldron as simple actors; but it is not similarly useful to describe such simple rules or actors as scientists in a research community.

suggests that the sprite or actor level is not the right level of abstraction for talking about intelligent interactions of concurrent agents. If multiple agents are to interact intelligently, they have to be complex enough to recognize WHY they need to interact. In simpler schemes, like schemas [Drescher86] or c-lines [Minsky77], there is no claim to "intelligent interaction" so that this problem does not arise. But when the agents of a model are deigned to be even slightly more complex, the problem of knowing "why" communication is necessary becomes critical.

Another important difference between actor schemes and cauldrons is that cauldrons-based reasoners have a more explicit control structure than typical actor-based systems. The application of a cauldrons-based reasoner to a problem is far more directed than the application of an actors-based system to the same problem. In a cauldrons-based reasoner, there is a clear notion of computational hierarchy, encompassing the relations between controllers and controlled. ***A cauldron is typically invoked by an executive rather than by a co-worker.***

Kornfeld's Ether [Kornfeld82] is a system very similar in design to cauldrons; it is a problem solving language which distributes computation over a collection of computational entities. In Ether, those entities are *viewpoints*,[9] while in my language, those entities are cauldrons.

While there are technical details of the Ether implementation that I have doubts about, such as its monotonicity or its separation of *sprites* (rules) and assertions, there are only two explicit differences between Ether and cauldrons which I would argue for:[10] one of these is a difference of intent; another is what I perceive as a serious problem with Ether's approach.

Cauldrons evolved from a collection of ideas about how the mind might work into a set of abstractions for building networks of "little minds" which perform reasoning tasks. As a result of this evolution, neither my implementation nor my theories contain notions like "computational power," since I cannot envison (possibly my own failing) a theory of mind which would support such a mechanism. Hewitt and Kornfeld in

9. Ether ***activities*** are the way that *computation* is divided, but ***viewpoints*** are the way *knowledge* is apportioned. Activities are used to parcel computational power, a mechanism I have chosen not to use, while viewpoints are used to parcel and restrict knowledge, which is the intent of cauldrons.

10. This means they were conscious decisions made in the development of both theories and implementation.

[Hewitt80] define a difference between their approach, "the scientific society" approach, and Minsky's "society of the mind/cognitive modeling" approach. The difference in intent between my work and Ether is essentially the same.

The one aspect of Ether which I have grave doubts about is the way in which programs in Ether are written. The standard way to solve a problem in Ether is to write a program which combines LISP code calling Ether primitives and Ether code calling LISP. Kornfeld stresses, in fact, that "Lisp should not be considered an 'escape mechanism' in Ether." My prime misgiving about this is that if a program must understand or be able to modify itself, having this mix of Lisp and Ether will make self modification very difficult.[11] This failing is demonstrated in Ether's separation of rules (sprites) and assertions; Ether sprites are written in Ether (a Lisp-like language), while Ether assertions are written in the language of the domain, with sprites only triggering off of assertions, not off of other sprites. While Ether is an excellent language for writing programs to solve any particular problem, when programs must modify either themselves or their approach to a problem, the difficulties of understanding the LISP code, and even worse, the interaction or the LISP and Ether code, will be hard to overcome.

Davis' meta-rules [Davis79], initiated much of my thought about strategies for restricting computations. The essence of meta-rules -- of having heuristics for heuristic selection -- is important in any situation where decisions about about the relevance of knowledge or methodologies are made. A range of issues that I have only tentatively explored revolves around this-- precisely, how do you choose which techniques to apply in solving a particular problem. It is likely that Doyle's [Doyle80] ideas about policy and intention will be an excellent starting point for systems which choose between multiple methods or approaches. The *need* mechanism in the example in the introduction is a first step in that direction. Another issue of interest is the means by which such meta-heuristics are acquired. (Lenat's recent work into heuretics [Lenat82] explores theories, models, and heuristics for thinking about heuristics).

An issue which meta-rules does not address is the issue of maintaining multiple instantiations of a reasoning program, where each uses different techniques and different knowledge. Davis's more recent work

11. While the implementation of a problem solving language may best be done in LISP (or some other traditional language), the inhomogeneity of mixing LISP and a problem solving language often cripples its reflexive flexibility, possibly leading it into the pitfall of "skill acquisition by DEFUN".

with Smith on **contract-nets** [Davis81a], does address this issue, but their approach differs from mine in the same way the as Ether's-- it seeks interesting and useful ways of distributing computation, rather than finding and testing theories which explain how the mind does things. Contract nets do capture the important concept of *distributed functionality*; a contract is an arrangement between different processes which are performing different tasks with different expertise. Further, Davis' latest work on multi-agent planning [Davis81b] considers the difficulties of multiple agents dealing with limited or unique resources in a coordinated way. These are issues which need far closer examination if distributed mechanisms are to be applied to real-world interactive problems.

The Hearsay systems [Lesser77] are architecturally similar to cauldrons. Cauldrons implement layers of abstraction in a reasoning program which can be compared to the levels of *hypothesis blackboards* in Hearsay. However, the locus of computation in cauldrons is different from that in Hearsay. In Hearsay, computation is performed by knowledge sources (KS's) which generate hypotheses on a given level of abstraction by examining adjacent levels. In cauldrons, the computation is centered in the cauldron rather than in the interaction between cauldrons. ***A cauldron, unlike a blackboard, computes as well as collects.***

Within each cauldron, generated hypotheses and plans are debugged and criticized by local procedures and knowledge, before being propagated up or down the abstraction hierarchy. This serves to hamper the propagation of bogus hypotheses between "blackboard levels". Additionally, the activities at any one level of the cauldron hierarchy are not homogeneous. The subcauldrons of a given cauldron apply a range of expertise and techniques to its subproblems or sublevels. Within a cauldron network, a range of "pockets of expertise" are dynamically created and destroyed as the system deals with new information or new hypotheses.

Many theories of distributed computation work with even smaller agents than actors. Schemes like the semantic network machines, constraints, or relaxation algorithms work with an agent size far smaller than cauldrons. The internal workings of a cauldron, with its interacting rules and assertions, may well be of comparable "grain size" to schemes like constraints or relaxation algorithms. It is on this level, and not in higher level parallelism, that I suspect the important efficiency gains will be made. I view higher level schemes like cauldrons as mechanisms for improving abstractions; lower level techniques like constraint propagation or marker passing I view as methods for improving efficiency.[12]

2.3 Static Cauldrons: A Note

The notion of cauldrons and frames described above arose from earlier work in learning and reasoning in highly parallel semantic networks. This work was begun in 1979 under Chuck Rieger at the University of Maryland and continued at MIT in 1980 and 1981. This work, unreported, began with a subset of Scott Fahlman's NETL [Fahlman79] (a highly parallel semantic network implementation) and extended the 'existence bit' of NETL to an 'existence word' for describing multiple belief or reasoning contexts. It then provided an 'implication link' for connecting statements (representational links in the semantic network). These implication links also had 'existence words' which affected the existence words they were able to propogate.

The static network so defined could entertain several states at once and each state (each bit of the allocated existence words) was called a 'cauldron.' As existence bits flicked to and fro throughout the network, the activity of the program proceeded along many lines or through many paths at once. The implementation of this network never got past a primitive simulation stage and eventually became (with the influence of other ideas at MIT) the set of ideas described here.

The implication links of the static implementation were used to activate patterns of 'rules' and 'assertions' in the network, and became the notion of frames (and of goal nodes) in the current implementation. One novel component of this activation notion was a selective version of the automatic 'frame creation' described at the end of the last chapter. New patterns of activation could be defined by capturing current patterns of activation, in particular by tracking the 'dependencies' for the current pattern of activation. All the activated nodes in the network, and the nodes which activated them, were used to specify a pattern of reactivation which could become a new 'frame' (in the parlance of the current implementation). Since reasoning was divided into levels by cauldrons, by constraining this recording to a single cauldron (a single bit in the propogated 'existence word') only immediately 'relevant' nodes would be recorded.

In retrospect (from 1986), those ideas seem worth returning to. In fact, promising systems like AFL

12. But of course, improving efficiency makes it possible to think about reasoning technologies which could never be seriously considered before.

[Blelloch86] seem to echo many of the ideas of this implementation. Furthermore, Agre's work on routine behaviour [Agre85] adopts the notion of inverting dependencies into activation patterns, though with a far more complete implementation.

3. Representation as Restriction

This section takes a teleological view of representation systems, working from the insistence that knowing what is *useful* is just as important as knowing what is *true*. A representation system does not merely provide a structure for making valid assumptions (defaults), it also provides a structure for determining what knowledge might be useful to a certain goal, in a certain situation, or for a particular inference.

The question of what knowledge is useful becomes relevant to the AI researcher only when one can choose what information ones program looks at when making inferences; precisely, when one has a mechanism for restricting consideration. Assertion based inference mechanisms without explicit control of focus and reference cannot offer a framework for establishing relevance-based representation schemes.

3.1 Knowledge Representation

This section describes a representation scheme working from the intuition that ***any knowledge representation is a paradigm for restriction of consideration.*** Taking this intuition, the solution of the representation problem becomes the space of solutions to the restriction problem. This gives a knowledge representation the clear design goal of *defining knowledge necessary for a particular performance.* This frame implementation is a mechanism for defining chunks of knowledge (collections of "active" rules and "inactive" statements) which may be mixed and merged to create tailored reasoning contexts.

The active and *procedural* nature of this tailored knowledge allows "meaning" to become an interpretation[13] of the computational context (the cauldron and the "chunks" of knowledge which define it) rather than any absolute definition. Making the knowledge an active part of the reasoning process lends power to these chunks of knowledge, allowing the knowledge representation to *implement* an epistemology, rather than merely fulfilling one.

13. In a very real sense of "interpret."

The basic unit of the representation is the *frame*, an assertion consisting of three parts:

o A *tag*, which is an identifier for the frame.

o A set of *inclusions*, each of which is the *tag* of another frame in the same cauldron. A frame is said to "virtually include" the frames attached to its inclusions.

o A set of *features, each of which is either a frame or an arbitrary object in the language of the problem solver.*

There are three basic operations on a frame:

o Feature alteration. The addition or removal of an object (or frame) to or from the features of the frame.

o Inclusion alteration. The addition or removal of a frame-tag to or from the inclusions of the frame.

o Frame Invocation. The "activation" of the frame into a problem solving context (cauldron). When a frame is "activated" into a cauldron, its *features* are asserted into the cauldron and each of its *inclusions* are activated, recursively, into the same cauldron.

A frame is defined as a set of arbitrary assertions (features) which may include, virtually, the assertions of any collection of other frames. Spreading activation through these inclusions then determines the set of assertions which are eventually added to the cauldron a frame is invoked into. These *features* (which may be rules, censors, arbitrary statements, or other frames) interact with other similarly activated features in the cauldron to perform tasks, complete inferences, or organize existing knowledge into new structures. The current implementation of this system has the bug that features are given precedence over inclusions by the mechanism of asserting the contents of inclusions before the features, allowing the immediate features to "clobber" elements of the inclusions. The exact problem is that since there is no implicit dependency maintenance accompanying non-monotonic modifications to the cauldron, "clobbering" does not always have the complete and correct effect. One way that this has been patched is to define rules -- in some particular reasoning contexts -- which keep track of the addition and deletion of assertions in a cauldron.

A key feature of this implementation is that it leaves all epistemological issues, such as the meanings of `A-Kind-Of` or `Part-Of`, to be *implemented in* the representation rather than being *defined by* the representation. It provides only a facility for defining chunks of knowledge to be applied and used by a program, leaving all issues of meaning and epistemology to interpretive structures implemented within the facility.

For instance, A-Kind-Of could be implemented as below:[14]

```
Frame
  Name: Class
  Inclusions: None
  Features:
    If  {class HAS-QUALITY CLASS}
        {instance IS-A-KIND-OF class}
      Then:
        If {class some-relation some-quality}
           NOT{instance some-relation another-quality}
           Then:
           {instance some-relation some-quality}
```

so that if there is a frame for carburetor:

```
Frame
  Name: Carburetor
  Inclusions: None
  Features:
    {Carburetor HAS-COLOR Black}
    {Carburetor HAS-PART  Foobar-Valve-Part}
          .
          .
```

14. The "code" given here is a pretty printed form of the actual assertions and rules in the current implementation. A frame has its parts clearly labeled; An IF form asserts those assertions after its THEN for each set of assertions matching the patterns before the THEN. Italicized words correspond to variables which will match anything and bold italicized variables refer to variables which have already been matched (and thus have values constrained by their earlier appearance). The first piece of code implements inheritance of defaults along AKO links. The frame CLASS contains a rule which says:

"If an *instance* is a kind of some class *class*, and *class* has a relation *relation* to some *value* which the *instance* does not explicitly have, then the *instance* has the same *relation* to the same *value*."

it is then possible to construct a frame for carburetor class:

```
Frame
  Name: Carburetor-class
  Inclusions: Carburetor, Class
  Features:
    {Carburetor HAS-QUALITY Class}
```

and for a particular carburetor:

```
Frame
  Name:  Carburetor-1
  Inclusions: Carburetor-class
  Features:
    {Carburetor-1 AKO Carburetor}
    {Carburetor-1 HAS-COLOR Silver}
```

which will have a Foobar-Valve-Part and would be black if it weren't clearly silver. The definition of *Carburetor-class* could also complain if some of the features of a given carburetor were undefined or inconsistent. For instance, if we could expand the definition of ***Carburetor-class*** to be:

```
Frame
  Name: Carburetor-class
  Inclusions: Carburetor, Class
  Features:
    {Carburetor HAS-QUALITY Class}

    If {instance AKO Carburetor}
       {instance HAS-COLOR Black}
      Then:
       {instance HAS-AGE Old}
       If {instance HAS-AGE New}
         Then:
          Error{A Carburetors age doesn't match its appearance.}
```

so that an "error report" would be generated (and presumably noticed) if a carburetors apparent condition did not match its given age.

We can also define variant forms of inheritance from classes or between instances (for there is much beyond AKO). For example, consider this definition of HAS-PART:

```
Frame
  Name: Part
  Inclusions: None
  Features:
    If {part HAS-QUALITY PART}
       {super-part HAS-PART part}
       {part HAS-PART sub-part}
      Then:
       {super-part HAS-PART sub-part}

    If {whole HAS-PART part}
      Then:
       {part IS-CONTAINED-IN whole}
```

Using the definition of part, it is possible to construct a frame for carburetor-part:

```
Frame
  Name: Carburetor-part
  Inclusions:  Carburetor-class, Part
  Features:
    {Carburetor HAS-QUALITY Part}
```

and for Carburetor-1:

```
Frame
  Name:  Carburetor-1
  Inclusions:  Carburetor-part
  Features:
    {Carburetor-1 AKO Carburetor}
    {Carburetor-1 HAS-COLOR Silver}
```

such that we may define this frame for Engine:

```
Frame
  Name: Engine
  Inclusions: Class, Carburetor-1
  Features:
    {Engine HAS-COLOR Blue}
    {Engine HAS-PART Carburetor-1}
        .
        .
        .
```

which has a number of interesting features, including possession of a `Foobar-Valve-Part` inherited from the carburetor via `Carburetor-1`. But this inheritance is *filtered inheritance* since other features of the carburetor, such as its color, are not inherited. This definition of `HAS-PART` is certainly weak and incomplete, but its definition reveals two powerful capabilities of this approach:

> Arbitrary defaulting or inheritance mechanisms may be defined explicitly and extensibly. These defaulting mechanisms need not merely default by inheriting through some hierarchy or heterarchy, but can refer to any elements of the current reasoning context.
>
> The semantics of a representation -- the way attributes are defaulted and constrained -- may be globally or locally redefined for a given reasoning context or a collection of reasoning contexts sharing a common *frame*.

If the language of the problem solver -- in which the contents of frames are written -- has assertions which can override or inhibit other assertions, (i.e. is non-monotonic) it becomes possible to wholly replace a definition of `AKO` or `HAS-PART` for a given frame. This permits the meaning of `AKO` or `HAS-PART` to vary depending on context, so that being `A-Kind-Of` "hacker" may use a very different interpretation of `A-Kind-Of` than being `A-Kind-Of` "heavy object".[15]

15. The introspective realization that hearing "Clyde is an elephant" immediately conveys a great deal of information about Clyde suggests that AKO is indeed special in some deep way. But the description "Clyde is elephant-shaped" invokes a large number of facts just as quickly, but leaves behind, *filters*, a great deal of non-shape information. Our recognition of this sort of simple analogical description (Clyde is shaped like an elephant) seem just as quick and natural as our recognition of instance-class descriptions, suggesting that AKO is not especially primordial.

3.2 Representation as Restriction: Other Approaches

The approach of implementing representational mechanisms like AKO in a general declarative language is in essence the same approach presented in [Hayes77] and [Nilsson80]. Both of these presentations describe skeletal frame implementations implemented in FOPC. In [Hayes77] Hayes further recognizes the utility of *filtered inheritance* -- described above -- as *seeing as*; for instance, seeing Thomas Jefferson *as* a scientist, rather than *as* a politician or *as* a slaveowner. But while simply embedding FRL or KRL in FOPC is interesting in itself, the result provides little of the real functionality in these languages. An FRL in FOPC may tell you what is *true*,[16] but it does little in the direction of telling you what is *useful*. Because cauldrons implement a notion of *local* knowledge, the measure of a rule or assertion's **relevance** suddenly gains meaning and importance. The frame implementation presented above organizes knowledge into ***contours of relevance***," fulfilling the requirement that a representation classify what is "useful" with the same facility that it classifies what is "true".

The chunking of knowledge into contours of relevance bears similarity to the use of the **theory** construct of Weyrauch's FOL [Weyrauch78] and its close descendant SDL (Structure Description Language) [Doyle80]. Both of these systems implement the concepts of structure and model defined for first order logic. A theory in FOL or SDL consists of three elements: a ***language*** defining the syntax of statements in the theory; a set of ***axioms*** which can generate valid statements in the theory; and a ***simulation structure*** which connects statements in the theory to some process which allows the statements to be interpreted and simplified. As with the frames containing a definition of AKO above, a theory is a collection of statements accompanied by a description of their syntax and semantics.[17]

The **theory** of FOL or SDL and the **frame** of my representation both provide mechanisms for chunking knowledge into contours of relevance, a feature lacking in Hayes and Nilssons examples. But all these schemes define only incomplete chunks of knowledge, and any reasoning process will consist of the interaction and intermixing of many such chunks. The mechanisms by which this intermixing is performed

---

16. Barring the monotonicity problems which keep FOPC distinguishing what was was once true from what is true. Doyle [Doyle80] discusses a flaw in default handling, the *family resemblance problem*, which arises from this monotonicity.

17. The simulation structure is used to give a theory *meaning*, a link between the theory as a representation and that which it is representing. In order for the sentence "s=1/2*a*t↑2" to have meaning, its elements {s,a,t,2,1/2,*,↑,=} must be attached to either arithmetic primitives or better, to a theory of equations in the reals.

are as important as the organization of the chunks themselves.

In FOL, the primary mechanism for theory interaction is *reflection.* Reflection allows statements in one theory to talk about theoremhood in another theory. Thus, if a meta-theory M talks about some theory T, and we wish to prove some theorem in T, we may be able to immediately generate the theory by performing some simple transformation in M. In FOL, reflection is invoked via the REFLECT command, which applies some principle from the meta-theory of the current theory to a set of statements in the current theory. One shortcoming with this mechanism is that though a theory can refer to its meta-theory, the meta-theory cannot refer to the component theories it describes. For instance, while a theory of peano arithmetic may reflect to the theory of reals which contains a summary of it, the same theory of reals cannot refer to peano arithmetic to justify new principles or axioms.

By contrast, in a cauldron hierarchy, higher levels invoke lower level activities to satisfy their plans and goals in addition to resolving conflicts and difficulties at the behest of lower levels. Thus the link between an active theory (cauldron) and the active theory which uses that theory (its parent) works both ways.

Doyle's SDL replaces reflection with a mechanism allowing semantic attachment to other theories as well as to implementation primitives. Thus, a theory can refer to another theory. Attachment to theories is used in concert with two other mechanisms, *virtual copies* and *typed parts.*

Virtual copies are a mechanism for inheriting assertions between theories. If a theory I copies a theory C, all the statements of C become *included* in I. This is identical to the *frame inclusion* mechanism used with cauldrons. Class instantiation is performed using virtual copies, by copying the theory of the class into the theory of the instance, and then attaching a constant in the instance to the global name of the theory of the class. In a cauldrons-based frame representation, the mechanism for instantiation is similar, but with the latter step, attachment, performed by explicit rules and axioms implementing some instantiation mechanism. Thus certain objects or techniques may perform this attachment in a completely different manner.

Typed parts are similar to virtual copies in that they copy another theory into the current theory. The difference is that the constants of the copied theory are replaced by *pathnames,* allowing a theory to have several distinct parts which instantiate the same theory. I argue that this typed part mechanism is misguided in haphazardly colliding sub-theories within their super theory. The point here is the subtle but important difference between *referral* and *containment.* A theory of real numbers may *refer* to a theory of peano arithmetic, but it will not *contain* that theory. The difference is that referenced sub-theories are instantiated separately from the theories which use them for justification or motivation. Cauldrons serve as a mechanism for instantiation which allows the separate instantiation of connected theories. A cauldron reasoning about a bicycle will generally invoke another cauldron to reason about the ball bearings in one of its wheels.

Agre and Chapman [Agre&Chapman83] replace Doyle's typed part mechanism with a mechanism called *virtual inclusion.* Virtual inclusion is essentially an extension of Doyle's virtual copy mechanism which allows the axioms of a copied theory to have their elements renamed to references in the theory being copied into. As with the typed part mechanism, this violates the modularity of knowledge -- the contours of relevance -- which the theory construct provides.

•

Since FOL is put forth as an interactive proof checker, there is little chance to "see it in action" with traditional AI tasks involving planning or reasoned deliberation. Weyrauch does not present cases where FOL chooses to use a particular theory, or decides to apply one theory in favor on another. FOL also avoids the problems of mixing theories (outside of the meta-theory attachments outlined above) with one and other by not attempting to implement any sort of inclusion or theory-copying mechanism. As described above, SDL attempts to broach these issues with a virtual copy mechanism, but needs to push the mechanism too far[18] due to the lack, I argue, of a way to explicitly use one theory from another theory without violating the principles of abstraction and modularity which separate theories from each other.

Regardless, given these tools for theory interaction, SDL attempts to attack more general AI issues, but its presentation in [Doyle80] is weak because Doyle uses essentially the same examples which Weyrauch presented in [Weyrauch78]. While Doyle speaks of reasoning about plans and sequences of actions (where actions are presumably attached to either other plan-theories or physical or mental acts), he never presents working examples which detail that reasoning in action. This may be because Doyle's definition of "mental acts" is no clearer than his specification of "physical acts," due to the absence of any "attention mechanism" with which to perform mental acts. Doyle does make a large number of important points about reasoning about reasoning, approaching such issues as how to decide which theory to deliberate with, how to modify a theory with respect to its success or failure, or whether to bother deliberating at all. Unfortunately, Doyle fails to ground this reasoning about reasoning to any real pieces of reasoning technology.

18. This is a common pitfall of representation systems. The AKO link in FRL [Roberts77] has the identical failure mode, inheriting all sorts of things and being used for a range of relations it had no business trying to express.

FOL and SDL are far more cleanly and precisely defined than cauldrons, but they leave enough questions unanswered to beg their sufficiency. Eventually, I can imagine cauldrons maturing into something as precisely defined as FOL, but currently the issues which cauldrons attempts to deal with, such as explicit attention control or "contours of relevance," are still too imprecise to be clearly and formally defined.

From the standpoint of 1986, we can note the development of SPHERE [Fillman83] as a version of FOL with more extensive multiple context reasoning facilities. We also see the extensive work of Genesereth and his colleagues [Genesereth82] as following the lines of explicitly describing how assertions and statements should be intererpreted and implemented.

3.3 Frames and K-Lines

Minsky's Society of the Mind offers a number of attractive mechanisms for restricting the computation involved in resolving of any particular task. The Society of the Mind offers a view of the mind as a collection of computationally simple agents communicating by channels called *c-lines* [Minsky77]. The composite of all the agents states, expressed as the states of their c-lines, defines the Society's, and hence the mind's, "mental state." Similarly, a *partial mental state* is the state of a subset of the minds c-line communication channels.

In [Minsky79], Minsky proposes that the mechanism of memory is one of *reactivating partial mental states.* When recalling an event, a fact, or a technique, a set of agents and c-lines are "activated" to interact with the current mental state. Minsky proposes that a partial mental state is stored by connecting the elements of the state (the c-line connections) to a *goal-node* by a set of *k-lines.* This connection is arranged so that if the goal-node is ever activated, the "activation" propagates through the k-lines to reactivate a stored partial mental state. A powerful addition to this enables k-lines to attach goal-nodes to other goal-nodes as well as to individual c-lines, thus creating a network which reflects a "hierarchy of activation". This hierarchy serves as a simple abstraction mechanism which can be expanded into representations for knowledge which may be logical, frame-like, or procedural.

K-Lines are constructed by goal-satisfaction; when a goal is satisfied, the the agents active on its satisfaction are recorded as an activation pattern. The links of this activation pattern --- from goal to agent --- are called k-lines. When a partial state of agents is reactivated through stored k-lines, the activation brings to bear the same agents, attitudes, and knowledge that were previously available when the goal was satisfied. One way of illustrating this idea is with the following image:

Imagine a large workshop with a wide variety of different tools easily available. The task of a workman is *to fix a broken bicycle, which is brought in and placed on a workbench draped with cloth. The workman begins working on the bicycle, using tools from around the workroom--- he is not too experienced with bicycle repair, so much of his work is trial and error, perhaps mixed the general ideas and principles concerning "the right way to do things."*

If a tools fits and does its job, it is dropped on the clot draped table.

If it fails to work it is tossed away.

Finally, the bicycle is repaired (the workman can notice this). He takes it down, puts it away, and picks up the tool-covered cloth on the worktable, wrapping it into a bundle. He tags this bundle with a ribbon maked "BICYCLE REPAIR" and places it on a shelf. The next time a bicycle needs repair, this bundle is unfolded onto the work table. All the tools are tehre, and work proceeds, without the necessity for trial and error as before.

If the tools are actually magic and intelligent screwdrivers, wrenches, etc; i.e. agents which work by themselves and interact to repair the bicycle, then the workman becomes part of the tools, and all that need be done is to dump the bundle of "tools" onto the table with the bicycle.

The frames implementation described above derives much of its inspiration from the intuition of "knowledge as reactivation" which k-lines provides. I have simply replaced parts of the c-line network and hierarchy by the arbitrary objects of a problem solving language (assertions, rules, etc). The generally hierarchical structure of a network of cauldrons parallels the c-line control hierarchy[19] described in [Minsky77]. The most prominent feature of this is that it allows an analogue to Minsky's **level-band-principle**, which confines specific "learning events"[20] to a tight range of levels of the abstraction hierarchy; thus learning a new technique for doing proofs doesn't generally affect the way one holds a pencil. As in any sort of programming, preserving layers of abstraction makes debugging (as well as understanding!) far easier. ***Cauldrons implement these "layers of abstraction" in a declarative programming language.***

Minsky calls the elements which trigger the k-lines "goal nodes," a regrettable misnomer, for while they have a good deal to do with reacting to explicit goals, they are also a powerful mechanism for structuring

19. This was the "BUILDER-WRECKER" hierarchy. In the cauldron system, each level of abstraction influences and "inspires" (as opposed to "controls") the level below it.
20. Precisely, k-line attachments.

knowledge of the world. Hence, having taken the "activation" parts of the k-lines theory and lumping them into the frame mechanism described above, I simplify the notion of goal-node to be something which is triggered by the presence of a particular class of goals.

I retain the term goal-node because it fits so well, defining a structure which ties together the frames relevant to achieving a particular goal. The functionality of the goal-nodes described here is far simpler than the functionality of the goal-nodes described in [Minsky79]. My goal-nodes do not implement memory, they implement the application of memory to a problem.

The goal-node used with cauldrons consists of two parts: a *trigger-pattern* and a set of *associated frames.* When a goal appears which fires off the trigger-pattern, a cauldron is started trying to achieve that goal, and the associated frames of the goal-node are "activated" into the cauldron. The goal-node thus serves as the mechanism by which a computation is begun and its direction and area of consideration defined.

For instance, in the blocks-world, the goal-node for making one block support another looks like:

```
Goal-node
  Trigger-pattern: {BLOCKA Supports BLOCKA}
  Activated-frames:
     DEFAULT-MAKE-SUPPORT-TECHNIQUE    ; How to do it.
     BLOCKA                            ; Details of one block.
     BLOCKB                            ; Details of the other.
     BLOCK-PROTECTIONS                 ; Censors.
```

The goal node triggers chunks of knowledge relevant to a given problem. This triggered knowledge might range from actual techniques to accumulated censors to knowledge about the particular blocks involved.

Goal-nodes serve to define a *reasoning context* for a particular task. This context contains knowledge about the details of the situation (for instance traits of individual blocks), general constraints of the larger current reasoning context (for instance blocks which are *protected* and untouchable), as well as knowledge about how to achieve ones goals.

4. Conclusions

This paper has described and demonstrated a collection of abstraction mechnanisms for distributed reasoning systems. In conclusion, I will outline the approach which motivated these mechanisms and then criticize both the approach and the mechanisms it engendered.

4.1 Philosophy of Minds

Cauldrons are an implementation of distributed problem solving developed in parallel with an emerging theory of distributed cognition. I have approached the issue of distributed computation in an unusual way; instead of starting with multiple processes and then figuring out how to distribute computation among them to perform tasks in an "intelligent" or "efficient" manner, I begin by looking at mechanisms of thought and then use the technology of distributed processing to model those mechanisms. The difference between these two approaches is one of intent; I am interested in how minds work, and am using computation as a touchstone for my theories. Instead of thinking in terms of a "distributed theory of computation," I am trying to begin the definition of a distributed theory of *mind.*

This theory views the mind as a collection of interacting computational processes which generate the surface phenomenon of rational and irrational thought. These processes have the following features:

o They are **restricted.**

o They are **dynamically created.**

o They **communicate** with one and other.

Restriction of processes is a requirement that *no process have a global viewpoint.* A given "piece of knowledge" interacts only with those pieces of knowledge *local* to it. Pieces of knowledge may react to or produce other pieces of knowledge, but they may only react to or immmediately effect the knowledge involved in the same process. This notion of *restricted consideration* is fundamental to the definition of a "process".

Artifical Intelligence is often criticized for succeeding only by restricting the domain of reasoning to a single "micro-world" or "expert-domain"; the flaw in these criticisms is the assumption that the processes of human reasoning are not similarly constrained. Human beings restrict the domains of their problem solving in much the same manner as AI researchers restrict the domains of their programs problem solving. When the mind is examining a play of Shakespeare, it is not at the same time considering some aspect of the integral of natural log. This ***restriction of consideration*** plays a critical role in the problem solving process. The restriction of reasoning processes provides the technology of *contexts of consideration* as well as a *closed world assumption* for the reflexive examination of those processes.

These restricted reasoning processes are *dynamically constructed and discarded*, at need, as the reasoner encounters new problems, new situations or new distractions. The process structure this creates is not a fixed piece of reasoning hardware, but constantly changes as reasoning proceeds. This notion of restricted mental processes appeals to folk-psychological notions of "focus" or "context".

It is in the dynamic creation of these processes that large pieces of knowledge are mobilized into the pursuit of tasks and goals. Thus the constraint of *restriction* is implemented in this mobilization, making sure that the pieces of knowledge and procedure present in a process are precisely relevant to its function. An activity is thus instantiated with most of the knowledge, both procedural and declarative, neccessary to its performance.

Communication allows one process to influence the events in another process. The constraint of *restriction* demands that this communication be limited, and we accede to this by allowing communication only through explicit channels and generally using those channels solely for the communication of control information.[21]

The greatest failing of any restriction scheme is the unexpected-- if a restricted reasoning process runs into unforseen interference, unsatisfied prerequisites, or internal inconsistencies, it is generally unable to act because of its narrow focus. Communication allows a recourse from these situations, permitting a process to request that some other process remove interferences, satisfy prerequistes or even debug the troubled process! These requests may be either resolved directly by an existing process or may instantiate a new process with the expertise for resolving the specific quandry.

The characterization of reasoning processes described above is reflected in the abstractions this paper has presented. The notion of restriction is implemented by the cauldron, as a restricted arena for deduction and inference. The dynamic creation of reasoning processes is embodied in the ability of a cauldron to create

21. Requests and replies as opposed to techniques or factual details. (A reply may of course be a "factual" detail, but we constrain replies of this nature to be "expected" in the sense that some mechanism exists within the process for handling them.)

other cauldrons, and particularly, in the goal-node mechanism. And finally, the necessity of communication between reasoning processes is provided by the frame mechanism, permitting cauldrons to communicate and share information between themselves. This communication is not merely between concurrent processes, but between processes separated by time, speaking through the channels of memory.

4.2 Failings and Futures

This research, like any research in a new area, has obvious failings. Primarily, these failings lie in the realm of unanswered questions and unmotivated assumptions. This section will attempt -- in brief -- to both motivate some basic assumptions and to describe future research on the many questions that still remain.

Why does one even want parallelism?

Parallelism is a way of keeping things small. Having conventions for making parts of an AI system separate and parallel (with only a few clearly defined dependencies between them) provides "fissure points" for making big problems smaller. An earlier version of this paper had a whole section motivating -- with folk psychological arguments -- such mechanisms in the mind. Its main argument evolved around the inability of the human mind to function well on problems which demanded large arbitrarily connected collections of knowledge. This observation, coupled with observations of AI programs which seemed to flounder in search problems, led to the principles of the previous section.

Is it "easier" to program in cauldrons than in other languages?

Sadly, no-- the organization of programs and the extension of programs already written seems easier in cauldrons than in other declarative languages. This is possible because one can refer to whole approaches and mechanisms as single chunks of rules or assertions. Unfortunately, the language in which one writes these chunks -- a sort of poor man's AMORD -- is clumsy to use. The language has no TMS or arbitration mechanism and as a result over half of ones code is making sure that the other half doesn't fire off unexpectedly and cleans up after itself when done.

Shouldn't there be more complex interactions between cauldrons?

This research has failed to broach this question in more than cursory detail. Cauldrons should be able to argue, conspire, and commune with each other. Metaphors about scientific communities and legislative bodies may well be useful at this level of detail. The only control or resolution structure this paper has motivated has been little different from classical subroutine invocation-- the complexities and interactions within a given cauldron may well be as complicated as the political and social interactions within a given group of humans.

Is the notion of "a collection of self interpreting assertions" presented here as a *frame* equivalent to the traditional notion of frame?

The traditional notion of frames -- of objects with properties -- has a notion of identity, of the required propogation of side-effects, which is absent from the scheme presented here. I originally believed that this notion of identity was yet anotchr epistemological notion to be implemented explicitly by rules in individual cauldrons. A large protion of the rules written for cauldrons handled the propogation of frame modification from cauldron to cauldron. The utility and apparent universality of these mechanisms have brought me to believe that a notion of identity -- absent from the mechanism I have propsed --is neccessary in a represetnation. Frames are attractive because they correspond to the introspective notion of distinct objects of thought. The frames of today's AI researchers are akin to the *ideas* and *impressions* of modern (read: the last 300 years) philosophers. They are the objects of thought and perception to which we attribute properties and refer to by other properties.

What about parallel implementations?

As I mentioned before, I primarily consider cauldrons to be a mechanism for abstraction rather than problem distribution. However I have put some thought into possible schemes for parallelizing cauldrons. Cauldrons seem to fall on a level of concurrency somewhat below supercomputers like the Connection Machine [Hillis81]. I see cauldrons being best implemented on a collection of medium sized typical von-Neumann connected by a fast bus -- or in a pinch, a fast network.

The individual processors of this machine could range in scale from a fast 6800 to a current generation Lisp

Machine.[22] Each processor would probably contain between one and five cauldrons interacting largely -- if at all -- locally. In fact, most processors may contain just one cauldron with five connected cauldrons below it.

22. Such as a Symbolics 3600 or a Xerox Dorado.

Appendix I - References

[Agre&Chapman83]

Agre, P.E. and Chapman, D., **Virtual Inclusion**, MIT Artificial Intelligence Laboratory, Working Paper 250, September 1983.

[Agre85]

Agre, P.E., **Routines**, MIT Artificial Intelligence Laboratory, Memo 828, May 1985.

[Blelloch86]

Blelloch, G., **AFL-1: A Programming Language for Massively Concurrent Computers**, S.M. Thesis, MIT Artificial Intelligence Laboratory, 1986.

[Davis79]

Davis, R. **Meta-rules: Reasoning about Control**, MIT Artificial Intelligence Laboratory, Memo 576, 1979.

[Davis81a]

Davis, R. & Smith, R., **Negotiation as a Metaphor for Distributed Problem Solving**, MIT Artificial Intelligence Laboratory, Memo 624, 1981.

[Davis81b]

Davis, R., **A Model for Planning in a Multi-agent Environment: Steps toward principles for teamwork**, MIT Artifical Intelligence Laboratory, Working Paper 217, 1981.

[deKleer78]

de Kleer, J., Doyle J., Rich, C., Steele, G., & Sussman G., **AMORD: A Deductive Procedure System**, MIT Artificial Intelligence Laboratory, Memo 435, 1978.

[Doyle78]

Doyle, J., **Truth Maintenance Systems for Problem Solving**, MIT Artificial Intelligence Laboratory, TR-419, 1979.

[Doyle80]

Doyle, J. **A Model For Deliberation, Action, and Introspection**, PHD Thesis, MIT Artificial Intelligence Laboratory, 1980.

[Drescher86]

Drescher, G., **Genetic AI - Translating Piaget into LISP -**, MIT Artificial Intelligence Laboratory, Memo 890, 1986.

[Fahlman79]

Fahlman, S.E., *NETL: A System for Representing and Using Real World Knowledge*, MIT Press, 1979.

[Fillman83]

Fillman, R.E., Lamping, J., and Montalvo, F.S., *Meta-language and meta-reasoning*, Proceedings of IJCAI VIII (IJCAI-83), Karlsruhe, West Germany 1983.

[Genesereth82]

Genesrth, M., *An Overview of Meta-Level Architecture*, Proceedings AAAI-83.

[Hayes77]

Hayes, P.J., *The Logic of Frames*, Department of Computer Science, University of Essex 1977

[Hewitt76]

Hewitt, C., **Viewing control structures as patterns of passing messages**, MIT Artificial Intelligence Laboratory, Memo 410, 1976.

[Hewitt80]

Hewitt, C., Kornfeld, W., **The Scientific Community Metaphor**, IEEE Systems, Man, & Cybernetics, Volume SMC-11, No. 1, 1980.

[Hillis81]

Hillis, D., **The Connection Machine**, MIT Press 1985.

[Kornfeld82]

Kornfeld, B.A., **Ether: A Langauge for Distributed Problem Solving.** MIT Artificial Intelligence Laboratory, TR-666, 1982.

[Lenat82]

Lenat, D., *Heuretics: Theoretical and Experimental Study of Heuristic Rules*, Proceedings AAAI-82, 1982.

[Lesser77]

Lesser, R. & Erman, L., *A Retrospective View of the HEARSAY-II Architecture*, Proceedings IJCAI5, Cambridge, Ma., 1977.

[McDermott74]

McDermott, D. & Sussman, G., **The Conniver Reference Manual**, MIT Artificial Intelligence Lab, Memo 259a, 1974.

[Minsky74]

Minsky, M., **A Framework for Representing Knowledge**, MIT Artificial Intelligence Laboratory, Memo 306, 1974.

[Minsky79]

Minsky, M., **K-Lines: A Theory of Memory**, MIT Artificial Intelligence Laboratory, Memo 516, 1979.

[Minsky77]

Minsky, M., **Plain Talk about Neurodevelopmental Epistemology**, MIT Artificial Intelligence Laboratory, Memo 430, 1977.

[Nilsson80]

Nilsson, N.J., *Principles of Artifical Intelligence*, Tioga Press, 1980.

[Sussman76]

Sussman, G.J., *A Computer Model of Skill Accquisition*, American Elsevier, New York 1975.