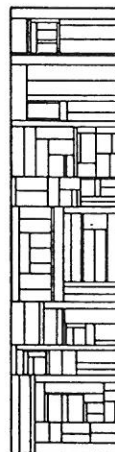


Free Blackboards and Anarchic Communities of Experts

Brian H. Mayoh

DAIMI PB - 207
March 1986

DATALOGISK AFDELING
Bygning 540 - Ny Munkegade - 8000 Aarhus C
tlf. (06) 12 83 55, telex 64767 aausci dk
Matematisk Institut Aarhus Universitet



PB - 207

B.H. Mayoh: Free Blackboards ...

TRYK: RECAU (06) 12 83 55

FREE BLACKBOARDS AND ANARCHIC COMMUNITIES OF EXPERTS

Abstract

The natural way of structuring a large expert system is to organize it as a collection of domain experts that communicate via a blackboard. We describe a "free" blackboard system that is being implemented on our Xerox 1108. A survey of the wide variety of languages and logics for knowledge representation and use motivates the freedom of the blackboard organisation: the blackboard must not restrict the kind of specialist domain experts that may be devised.

CONTENTS

1.	Free Blackboards	4
2.	Panorama of Specialists	8
2.1	Modules in conventional programming languages	8
2.2	Data bases as experts	10
2.3	Prolog experts	10
2.4	Abstract datatypes	11
2.5	Petri nets	11
2.6	Interface experts	12
3.	Logics, Heuristics and Strategies	13
3.1	Time	14
3.2	Space	15
3.3	Truth	16
3.4	Proof	20
3.5	Belief	21
3.6	Knowledge	22
4.	Dynamic Logics, Actions and Planning	23
4.1	Situations	24
4.2	Dynamic logic	25
4.3	Structured actions	26
5.	Implementation	29
	References	31

Free Blackboards and Anarchic Communities of Experts

If an expert system is to be useful, it must have a large knowledge base and this knowledge base must be structured. The natural way of structuring a large expert system is to organize it as a collection of domain experts that communicate via a blackboard. The blackboard idea was introduced in HEARSAY (EHR,LeEr) and there are many versions of the idea (NiAi,Hay,EnGo), but most versions place severe restrictions on the freedom of domain experts. The way a domain expert keeps and uses information should depend on the domain of expertise, and it should not be restricted by the blackboard information. In section 1 of this paper we describe a simple "free" blackboard organisation, that places no restrictions on how a domain expert keeps and uses its information. In section 2 we present a panorama of languages for domain experts: conventional programming languages, data bases, logical programming languages, abstract data types, re-writing rules, Petri nets, interface grammars. In section 3 we present a panorama of static logics for domain experts: temporal, spatial, fuzzy, multivalued, non-monotonic, belief and knowledge logics. In section 4 we present three planning logics for domain experts: Situational, dynamic, and action logics. In the last section we indicate how the free blackboard organisation can be implemented on the Xerox 1108 using LOOPS.

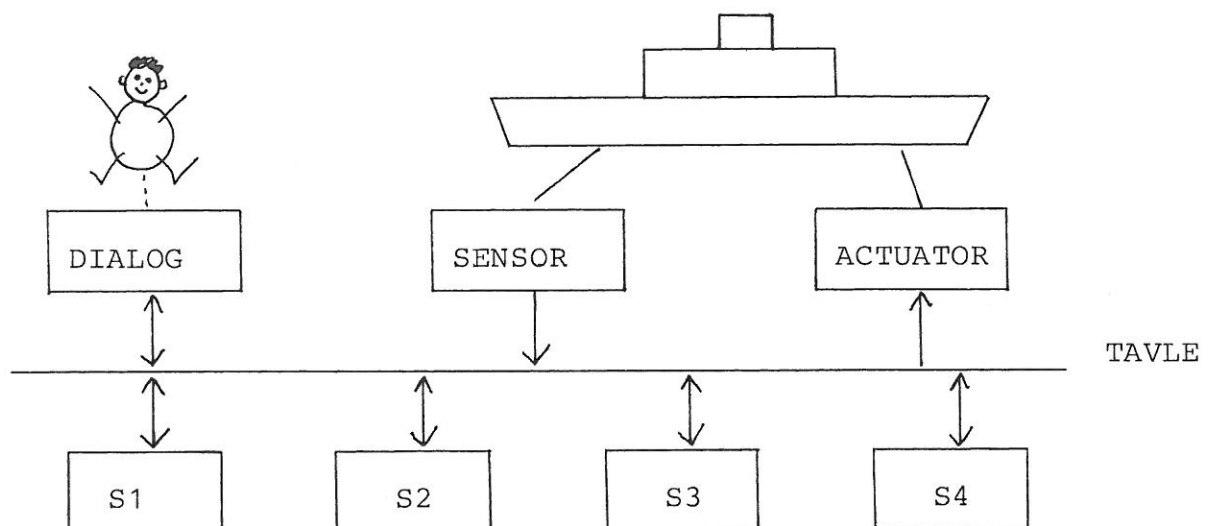


Fig. 1A Free Blackboard

#1 Free Blackboards

The natural way to organise a large expert system is as a community of domain experts, "specialists", that communicate via a blackboard. We will describe many kinds of domain experts and the many ways that can keep, use and communicate information. Internally domain experts vary greatly, and the motivation for the free blackboard organisation is to enforce an "external" uniformity on the variety of domain experts so that unruly members of the community do not lead to deadlock, lack of progress and other disasters.

Externally each domain expert is responsible for a number of relations. We distinguish between high level relations, which have relations as arguments, and low level relations, which do not. In later sections we will describe how one can provide many powerful and useful tools for domain experts with high level relations- It is not surprising that such tools can be provided, because there have been many advocates for "the power of matallevels" in the past few years (Bun, BoKo, Bow, FLM, Weh).

How does the free blackboard ask for information from a domain expert? One way is to send it a message with the external form

$? \quad R \text{ (arg, ... arg}_n\text{)}$

and expect one substituiton σ , such that "R holds for $\langle \sigma(\text{arg}_1) \dots \sigma(\text{arg}_n) \rangle$ ", as the reply.

Another way of asking for information is to send a message with the external form

$?? R (arg_1 \dots arg_n)$

and expect all substitutions σ , such that
 "R holds for $\langle \sigma(arg_1) \dots \sigma(arg_n) \rangle$ ", as the reply.

Later we will explain how the free blackboard handles unexpected replies to queries - no substitution σ for ? R messages, infinitely many for ?? R messages.

How does the free blackboard tell new information to a domain expert? The only way is to send it a message with the external form

$! R (arg_1 \dots arg_n)$

and expect it to rearrange its knowledge base so that "R holds for all substitution instances of $\langle arg_1 \dots arg_n \rangle$ ".

Domain experts can be like elephants - monotonic, unable to cancel, delete, annul or forget information - but they may be able to handle messages with the external form

$! \text{ Cancel } (R(arg_1 \dots arg_n))$

appropriately. In section 4 we will discuss domain experts that can execute more refined actions than just adding and deleting information. Such experts may be able to respond appropriately to messages with external form

! Execute (plan)

To explain the subtle distinction between "Cancel", "Execute" and normal relations we must look at the semantics of domain experts. At any given time a domain expert has a knowledge base K_0 and a logic L_0 . Implicitly the expert has the structure (theory, universe)

$$M_0 = \{\text{instances of its relations that can be inferred from } K_0 \text{ by } L_0\}$$

and its response to a message is determined by M_0 .

For each relation R , maintained by the domain expert, we have

$M_0 \upharpoonright R = \{\text{instances } R(a_1 \dots a_m) \text{ that can be inferred from } K_0 \text{ by } L_0\}$ and the complete replies to the messages $??R(a_1 \dots a_m)$ and $?S(b_1 \dots b_n)$ are

$$\{\text{substitution } \sigma \mid R(\sigma(a_1) \dots \sigma(a_m)) \in M_0 \upharpoonright R\}$$

and if $S(\sigma(b_1) \dots \sigma(b_n)) \in M_0 \upharpoonright S$ for some substitution σ

then any such σ else none

respectively. If the blackboard sends the message $!R(a_1 \dots a_m)$, then the domain expert must change K_0 to K_1 and L_0 to L_1 so that all substitution instances of $R(a_1 \dots a_m)$ can be inferred from K_1 by L_1 . For many messages $!R(a_1 \dots a_m)$ the domain expert need do no more than

$$L_1 = L_0 \quad K_1 = K_0 \cup \{\text{substitution instances of } R(a_1 \dots a_m)\}$$

but expert may have to be more subtle when R is a highlevel relation. If the blackboard sends the message $!Cancel(R(a_1 \dots a_m))$, then the domain expert must change K_0 to K_1 and L_0 to L_1 so that no substitution instance of $R(a_1 \dots a_m)$ can be inferred from K_1 by L_1 . For some messages $!Cancel(R(a_1 \dots a_m))$ the domain expert need do no more than

$$L_1 = L_0 \quad K_1 = K_0 - \{\text{substitution instances of } R(a_1 \dots a_m)\}$$

but this is insufficient when some substitution instances of $R(a_1 \dots a_n)$ can be inferred from K_1 by L_0 . Now suppose the black-board sends the message !Execute(Plan). For each action in the plan, the domain expert changes its knowledge base K and/or its logic L . Thus high level relations like "Cancel" and "Execute" are only used in !-messages to change the structure in a domain expert, other high level relations can also be used in ?- and ??-messages because they have instances in the structure of the domain expert.

Gallery comment (May 1)

The signature Σ of a domain expert is the collection of relations managed by the expert; more precisely, Σ has a name for each of these relations and sufficient "sort" information to fix the set of Σ -frames

$$\text{FRM}(\Sigma) = \{\text{well formed formulae using only relation symbols in } \Sigma\}.$$

The set of Σ -structures is defined to be

$$\text{STR}(\Sigma) = \{\text{subsets of SENT}(\Sigma)\}$$

where $\text{SENT}(\Sigma) = \{e \in \text{FRM}(\Sigma) \mid \text{no free variables in } e\}$.

The domain expert gallery is given by

$$\text{Val}_{\Sigma}(m, e) = \{\text{substitutions } \sigma \mid \sigma(e) \in m\}$$

for $e \in \text{FRM}(\Sigma)$ and $m \in \text{STR}(\Sigma)$. The complete response to the message ??e is the set $\text{Val}_{\Sigma}(m, e)$, when m is the current structure; if $\text{Val}_{\Sigma}(m, e)$ is empty, then none is a complete response to the message ?e; if $\text{Val}_{\Sigma}(m, e)$ is not empty, then any of its elements is a complete response to the message ?e. When a domain expert is sent a !-message it has to revise its current structure appropriately. For each such message !e we have a relation $\underline{\alpha}_{\Sigma} \subset \text{STR}(\Sigma)^2$ such that

$m_0 \xrightarrow{\alpha} m_1$ iff for some K_0, L_0, K_1, L_1 we have

$\langle K_0, L_0 \rangle$ gives m_0 & $\langle K_1, L_1 \rangle$ gives m_1 &

α changes $\langle K_0, L_0 \rangle$ to $\langle K_1, L_1 \rangle$

For the message !e we must have

$\sigma \in \text{Val}(e, m_1)$

for all substitutions for the message !Cancel(e) we must have

$\text{Val}(e, m_1)$ is empty

and there are no requirements on m_1 for the message !Execute(plan).

#2 Panorama of specialists

In expert systems one usually thinks of specialist, domain experts as collections of facts, rules and heuristics encoded in LISP, PROLOG or one of the many special programming languages for expert systems. This kind of specialist domain expert is described in #2.3. but it is only one of the many answers to the question "What kinds of specialists are useful in free blackboard expert systems"? Another answer is "specialists that use an analogy or diagram which they consult and change when they respond to messages". Let us look at some of the other answers.

#2.1 Modules in conventional programming languages

Most modern programming languages allow one to write modules with

- typed variables to keep values
- functions that return values when called
- procedures that change values when called

In Ada these modules are called "packages", in object oriented languages like Smalltalk these modules are called "classes". Whatever the language a module is programmed in, the programmed module is a specialist domain expert. If f is a function in a module M , then M can handle a message $?f(x_1 \dots x_n, x_0)$ by

- (1) rejecting the message if the actual parameters $x_1 \dots x_n$ do not satisfy the type and mode requirements for f
- (2) computing the value v_0 of $f(x_1 \dots x_n)$
- (3) returning $f(x_1 \dots x_n, v_0)$ as the reply to the message.

If R is a procedure in a module M , then M can handle messages $!R(x_1 \dots x_n)$ by

- (1) rejecting the message if $x_1 \dots x_n$ do not satisfy the parameter requirements
- (2) calling $R(x_1 \dots x_n)$
- (3) replying done

If T is a type of variables in the module M , then M can reply to messages $?T(a, v)$ with $T(a, v)$ where v is the value of the variable $a:T$ and it can handle $!T(a, v)$ messages by assigning v as the value of the variable a .

Records, arrays and other structured types are no problem - for each component s of T we have relation $T.s$

- the expert handles $!T.sel(a, v)$ by $a.sel := v$
- the expert handles $?T.5(a, v)$ by returning $T[a, a[5]]$

The advantage of program experts is efficiency. The disadvantage is higher order relations cannot only be captured with difficulty because programming languages treat procedures and functions as second class citizens.

#2.2 Data bases as experts

The conceptual schema of a data base gives a number of relations. The updating mechanism gives a way of handling !R-messages for any R in the schema, and it may give a way of handling !Cancel (R(...)) and !(Replace (R(...), R(...))) messages. The quering mechanism gives a way of handling ??Query (F(...)) messages where F() is a query built from relations in the data base scheme. In file systems we only have ??Query (R(...)) which is equivalent to ??R(..).

The disadvantage of database experts is that these are the only messages; the advantage is that they are efficient with large knowledge bases and the system can provide tools for handling queries and updates.

#2.3 Prolog experts

Any collection of PROLOG facts and rules can be considered as an expert system DE. Such a collection can handle any message whatsoever;

!R() is handled by adding R() as a fact
 !R() is handled by asking for one proof of R()
 ??R() is handled by asking for all proofs of R().

In practice Prolog experts should be responsible for one set Σ of relations R and the blackboard should reject R-messages to Prolog expert E when E is not responsible for R.

Example

Several biological expert systems know how to reverse DNA sequences (LHBG). This knowledge can be expressed in PROLOG as:

```
reverse (< >, < >).
reverse (<a>, <a>).
reverse (join(x,y), join(v,u)):- reverse(x,u) reverse(y,v).
join (x,join(y,z)):- join(join(x,y),z).
```

#2.4 Abstract Datatypes

Sometimes it is convenient to represent knowledge using equations. A collection of equations S can handle any message:

$!R(x_0 \dots x_n)$ is handled by adding $r(x_1 \dots x_n) = x_0$

$?R(x_0 \dots x_n)$ is handled by asking for a proof of $r(x_1 \dots x_n) = x_0$

$??R(x_0 \dots x_n)$ is handled by asking for all proofs of $r(x_1 \dots x_n) = x_0$.

A domain expert can have S as its knowledge base and it can derive new equations by substitution and perhaps by conditional rules. Software tools can provide efficient equation deriving techniques like narrowing (GoMe) and the theory of equation solving, abstract data types, is well developed.

Example

The knowledge of how to reverse DNA sequences can be represented as the abstract data type

```
reverse (< >) = < >
reverse (<a>) = <a>
reverse (join(x,y)) = join(reverse(y), reverse(x))
join (x,join(y,z)) = join(join(x,y),z).
```

#2.5 Petri nets

Sometimes it is convenient to represent knowledge in the form of a Petri net. A token with label $(a_1 \dots a_n)$ at a place R represents the fact $R(a_1 \dots a_n)$, so a marking of the net represents the knowledge base of a domain expert. Transitions in the net represent production rules, so an unmarked net represents the logic of a domain expert. Facts which are true for all possible expert markings can be represented by net facts (dead transitions, see fig. 2). Software tools can provide analysis and synthesis techniques for net experts (St.Th).

Example

In figure 2 we show how simple logical propositions can be represented as net facts - the strange boxes are dead transitions, transitions which never fire. Complex logical propositions tend to have simple net representations.

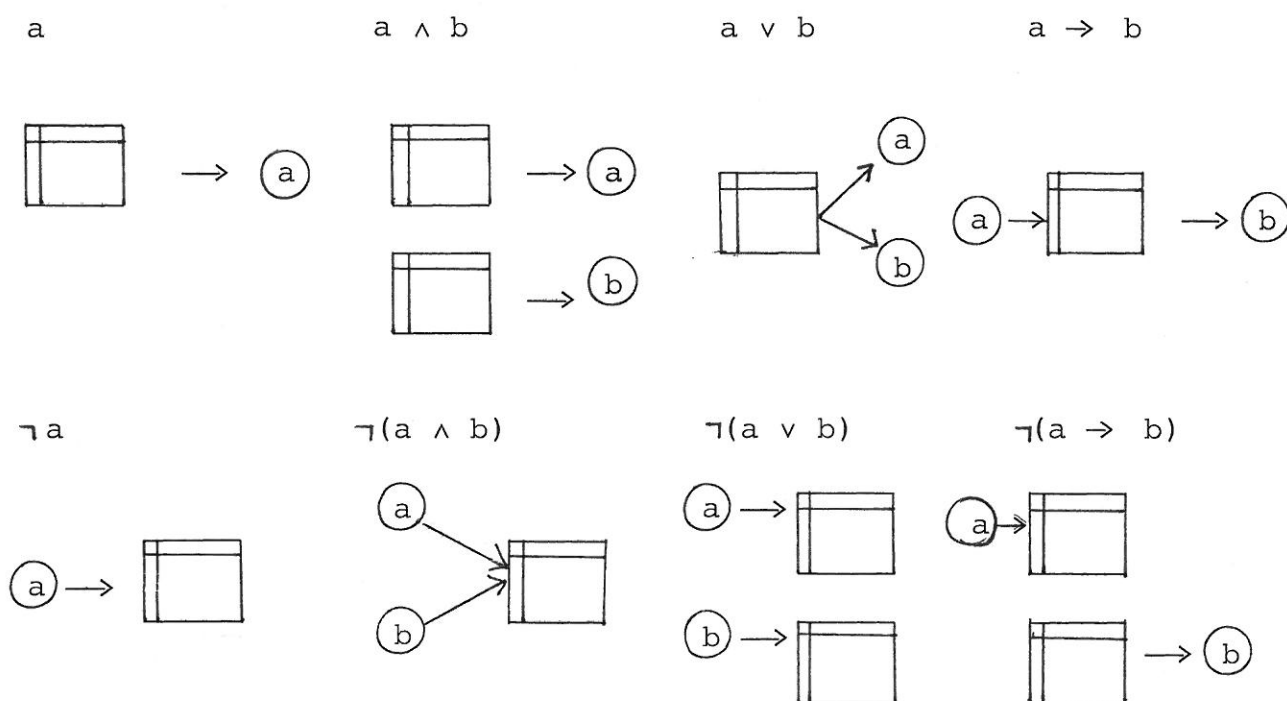


Fig. 2. Logical facts in Petri Nets.

#2.6 Interface experts

How does society of domain experts interact with people and mechanical devices in the real external world? There must be some interface experts in the community if the community is to be able to

- (1) communicate with external users whether in natural language or formal language
- (2) accept information from sensors (thermostats, cameras, etc.)
- (3) make changes in the external world by actuators (closing valves, lighting fires, etc.)

An interface expert will presumably send messages to other domain experts in the community when it receives information from a user or a sensor. An interface expert will occasionally communicate with a user or an actuator, when it is sent a message by other domain experts in the community (replies to interface expert questions are also messages). Much work on (1) has been done by database/expertsystem devisers and parsing language analysis can be supported by software tools.

#3 Logics, Heuristics and Strategies

Specialist domain experts can have their own logics for handling $!-,?-,??-$ messages. In #1 we said that a specialist has a knowledge base K_0 and a logic L_0 at any given time t_0 . Each specialist has an inference mechanism that determines a structure M_0 from a knowledge base K_0 and a logic L_0 . For the conventional language specialist of #2.1 the knowledge base K_0 is given by the values of variables and the logic L_0 is given by a collection of functions and procedures. For the database specialists of #2.2 the knowledge base K_0 is the stored data and the logic L_0 is given by "views" defining derived relations. For the Prolog specialists of #2.3, the knowledge base K_0 is a set of facts and the logic L_0 is given by a set of rules. For the datatype specialists of #2.4, the logic L_0 is given by equations and K_0 is empty. For the term rewriting specialists of #2.5 the knowledge base K_0 is a set of terms and the logic L_0 is given by term rewriting rules. For the Petri net specialists of #2.6 the knowledge base K_0 is a set of facts and the logic L_0 is given by a Petri Net. For the interface specialists of #2.7 the knowledge base K_0 is a set of "semantic" facts and the logic L_0 is given by translation rules.

Clearly our notion of logic is rather broad but it should be even broader - not only should it allow for "proof rules" but also for heuristics and strategies. The inference mechanism for a specialist domain expert should determine the structure M_0

as the extension of the knowledge base K_0 by applying the rules and heuristics of the logic L_0 . In the LOOPS inference mechanism rules are given in rule sets and the application of a rule set is guided by one of four possible heuristics. The inference mechanism for PROLOG and some other rule systems allows for more general heuristics by "going to the meta level". The basic idea is that a heuristic or strategy "if $P()$ then apply $L():-R_1()..R_n()$ " can be captured by the meta rule

$$H((L(), \dots):- P() H(R_1(), \dots) \dots H(R_n(), \dots))$$

This idea seems to come from Kowalski and it has become very popular; in the rest of this section we shall see why.

#3.1 Time

In many expert domains facts are not eternal truths, sometimes they hold and sometimes they do not. The facts in the knowledge base of a specialist in such a domain should be of the form

$$K(\text{formula}, t)$$

where t is either a time instance or a time interval. The logic for the domain specialist should allow arguments about time. There are many temporal logics available (All 1, All 2, AlHa, Bur 1, Bur 2, Hum, KaGo, Kon, Reur, Sho, Vil) and the specialist may decide to use a logic provided by a software tool.

Example

Every logic for time intervals has the rule

$$\text{During}(t_1, t_3) :- \text{During}(t_1, t_2) \text{ During}(t_2, t_3).$$

A specialist domain expert may well use a software tool to provide such temporal rules and general rules like

$$K(\text{formula}, t_1) :- K(\text{formula}, t_3) \text{ During}(t_1, t_3)$$

that connect specialist knowledge with temporal rules.

#3.2 Space

In many expert domains facts are not global truths, they hold at some regions and points in space and not at others. The facts in the knowledge base of a specialist in such a domain should be of the form

$$K(\text{formula}, s, \dots)$$

where s is either a point or a region in space. The logic for the domain specialist should allow arguments about space. Several spatial logics are available (MaBi, ReSi) and the specialist may decide to use a logic provided by a software tool.

Example

Every logic for spatial regions has the rule

$$\text{Within}(s_1, s_3) :- \text{Within}(s_1, s_2) \text{ Within}(s_2, s_3).$$

A specialist domain expert may well use a software tool to provide such spatial rules and general rules like

$$K(\text{formula}, s_1, t_1) :- K(\text{formula}, s_3, t_3) \text{ Within}(s_1, s_3) \text{ During}(t_1, t_3).$$

that connect specialist knowledge with temporal and spatial rules.

#3.3 Truth

In many expert domains facts are not absolute truths, they can be imprecise and uncertain. The facts in the knowledge base of a specialist in such a domain should be of the form

$$K(\text{formula}, v, \dots)$$

where v is some kind of "truthvalue". The logic for the domain specialist should allow arguments about truth values, in particular it should define the truth value of a compound formula from the truth values of its components. Many kinds of exotic truth values have been used in existing expert systems: uncertainty factors, probabilities, possibilities, and "fuzzy values". (Doy, FaPr, McC2, McDo, MoD2, OrPa, Prd, Rei, Zad). In such an expert system we can have rules like

$$(*) \quad K(F_0, f(v_1 \dots v_m) \dots) :- K(F_1, v_1 \dots), \dots, K(F_m, v_m \dots).$$

$$(**) \quad K(F_1 \& F_2, (v_1 \& v_2, \dots)) :- K(F_1, v_1 \dots) \ K(F_2, v_2 \dots)$$

Note that logical operators like $\&$ in $(**)$ can be used ambiguously both as semantic functions for combining truth values and as syntactic ways of building a complex formula from simple components. This duality is so attractive that the truth values in most exotic logics form a "pseudo" Boolean algebra and all expert using the logic have syntactic variants of the rules

$$K(F_1 \wedge F_2, v_1 \wedge v_2 \dots) :- K(F_1, v_1 \dots) \ K(F_2, v_2 \dots).$$

$$K(F_1 \vee F_2, v_1 \vee v_2 \dots) :- K(F_1, v_1 \dots) \ K(F_2, v_2 \dots).$$

$$K(F_1 \Rightarrow F_2, v_1 \Rightarrow v_2 \dots) :- K(F_1, v_1 \dots) \ K(F_2, v_2 \dots).$$

$$K(F_1 \Leftrightarrow F_2, v_1 \Leftrightarrow v_2 \dots) :- K(F_1, v_1 \dots) \ K(F_2, v_2 \dots).$$

$$K(\neg F, \neg v \dots) :- K(F, v \dots).$$

For classical logic these rules can be

$$\begin{aligned}
 K(F_1 \wedge F_2, \min(v_1, v_2) \dots) & \quad :- K(F_1, v_1) \ K(F_2, v_2). \\
 K(F_1 \vee F_2, \max(v_1, v_2) \dots) & \quad :- K(F_1, v_1) \ K(F_2, v_2). \\
 K(F_1 \Rightarrow F_2, \max(1-v_1, v_2) \dots) & \quad :- K(F_1, v_1) \ K(F_2, v_2). \\
 K(F_1 \Leftrightarrow F_2, 1-\max(v_1 \cdot v_2, v_2 \cdot v_1)) & \quad :- K(F_1, v_1) \ K(F_2, v_2). \\
 K(\neg F, 1-v \dots) & \quad :- K(F, v).
 \end{aligned}$$

where the values 1 and 0 correspond to true and false respectively.

Recently there has been much discussion of incomplete and inconsistent knowledge bases, PROLOG's negation as failure, truth maintenance systems and non-monotonic logic. When truth values are explicit, one gets a clearer view of the problems. Intuitively there are several notions of when a knowledge base K_0 and a logic L_0 are incomplete:

- (1) there is a formula F such that $K(F, v) \notin K_0$ for all v
- (2) there is a formula F such that $K(F, v) \notin M_0$ for all v
(ie no $K(F, v)$ can be derived from K_0 by L_0)
- (3) there is a formula F such that $K(F, v), K(\neg F, v) \notin K_0$ for all v
- (4) there is a formula F such that $K(F, v), K(\neg F, v) \notin M_0$ for all v .

The "negation as failure" convention rules out

- (3) -incompleteness by: $K(F_1, \underline{\text{false}}) \notin K_0 \equiv K(F, \underline{\text{false}}) \notin K_0' \vee \text{no } K(F, v) \notin K_0'$
for some K_0' .

However there is no good reason to adopt such conventions; specialist domain experts may well be incomplete. For each notion of incompleteness there is a notion of when a knowledge base K_0 and a logic L_0 are inconsistent:

- (1') there is a formula F such that $K(F, v_1), K(F, v_2) \in K_0$ for $v_1 \# v_2$
- (2') there is a formula F such that $K(F, v_1), K(F, v_2) \in m_0$ for $v_1 \# v_2$
- (3') there is a formula F such that $K(F, v_1), K(\neg F, v_2) \in K_0$ for $v_1 \# \neg v_2$
- (4') there is a formula F such that $K(F, v_1), K(\neg F, v_2) \in m_0$ for $v_1 \# \neg v_2$

where $v_1 \# v_2$ abbreviates "truth values v_1 and v_2 are incompatible".

A specialist domain expert is a "truth maintenance system" if it tries to keep its knowledge base K_0 and its logic L_0 consistent. Ideally such a specialist should check for consistency when it is sent a !-message, before making the appropriate changes to its knowledge base and logic. In practice the specialist can allow temporary inconsistency and make its knowledge base and logic consistent when it has free time, when it is not busy replying to messages. The basic idea behind some truth maintenance systems is that the facts in the knowledge base K_0 are of the forms

$$K(F, \underline{\text{in}}) \qquad K(F, \underline{\text{out}})$$

the logic L_0 can only derive in-facts, and consistency is maintained by converting in-facts to out-facts.

In such a system we can represent K_0 as a "four-truth valued knowledge base K_0' "

$$\begin{aligned}
& K(F, \underline{\text{in}}), K(\neg F, \underline{\text{in}}) \in K_0 \Rightarrow K(F, \underline{\text{contradiction}}), K(\neg F, \underline{\text{contradiction}}) \in K'_0 \\
& K(F, \underline{\text{in}}) \in K_0, K(\neg F, \underline{\text{in}}) \notin K_0 \Rightarrow K(F, \underline{\text{true}}), K(\neg F, \underline{\text{false}}) \in K'_0 \\
& K(F, \underline{\text{in}}) \notin K_0, K(\neg F, \underline{\text{in}}) \in K_0 \Rightarrow K(F, \underline{\text{false}}), K(\neg F, \underline{\text{true}}) \in K'_0 \\
& K(F, \underline{\text{in}}), K(\neg F, \underline{\text{in}}) \notin K_0 \Rightarrow K(F, \underline{\text{undefined}}), K(\neg F, \underline{\text{undefined}}) \in K'_0
\end{aligned}$$

and the logic L_0 can be converted to a four-valued logic L'_0 .

There is a close connection between truth maintenance systems (Goo, McD2) four valued logics (San) and non-monotonic logics. A specialist domain expert uses non-monotonic logic if it can have knowledge bases K_0, K_1 and logics L_0, L_1 such that $K_0 \subseteq K_1$ and $L_0 = L_1$ but not $M_0 \subseteq M_1$. Non-monotonic logics, like fuzzy logics, are powerful ways of arguing about uncertain and imprecise knowledge. Specialist domain experts may well decide to use an fuzzy or non-monotonic logic provided by a software tool.

Comment

We have discussed how the truth value of a formula F may be independent of the truth value $\neg F$. For a discussion of the distinction between uncertainty and imprecision see (Prd); "Population (Århus) < 1000000" is certain but imprecise, "Population (Århus) = 150000" is precise but uncertain, both precision and certainty can be reflected in truth values.

Comment

The unified theory of many areas of computer science, being developed by the author (May 1), the key notion is a gallery, a function

$$G: \text{Structure} \times \text{Frame} \longrightarrow \text{Value}$$

indexed by a set of signatures. Every gallery gives a specialist domain expert for each signature Σ by

- F is a domain formula $\equiv F \in \text{frame}(\Sigma)$
- K_0 is possible knowledge base $\equiv K_0 \subseteq \{K(F,v) \mid G(M,F) = v\} \quad m \in M$

The semantics of every domain expert is given for some $m \in \text{Structure}(\Sigma)$ a gallery.

#3.4 Proof

In many expert domains imprecise and uncertain facts may be supported by arguments or proofs from assumptions. The facts in the knowledge base of a specialist in such a domain should be of the form

$K(\text{formula}, p, \dots)$

where p is some kind of proof, argument or endorsement. The logic for the domain specialist should allow arguments about proofs, in particular it should define the proof of a compound formula from proofs of its components. Some appropriate logics are implicit in recent expert systems - eg (Coh) - and professional logicians have suggested others: intuitionistic type theory (see May 2) and logics for provability (Boo). In such a logic one might have syntactic variants of the rules

$K(F_1 \wedge F_2, p_1 \wedge p_2) :- K(F_1, p_1) \quad K(F_2, p_2).$

$K(F_1 \vee F_2, \text{first}(p)) :- K(F_1, p).$

$K(F_1 \vee F_2, \text{second}(p)) :- K(F_2, p).$

$K(F_2, \text{mp}(p_1, p_2)) :- K(F_1, p_1) \quad K(F_1 \rightarrow F_2, p_2).$

A specialist domain expert may well decide to use a proof logic provided by a software tool. One can buy many expert building systems which have a tool for generating explanations from proofs eg. LOOPS provides proofs in the form of "audit trails".

Notice that specialists can learn from their experience when proofs are explicit. Notice also that explicit proofs give a natural way of revising the knowledge base and logic of a specialist domain expert when it discovers an inconsistency or it receives a !-message that is incompatible with its current knowledge base K_0 and logic L_0 . Clearly there is a close connection between explicit proofs and non-monotonic logic.

#3.5 Belief

In many expert domains facts may only be beliefs of particular individuals or common beliefs. The facts in the knowledge base of a specialist in such a domain should be of the form

$$K(\text{formula}, \mathcal{X}, \dots)$$

where \mathcal{X} is some kind of "believer". The logic for the domain specialist should allow arguments about beliefs, in particular it should allow for self belief, embedded belief - $K(\text{formula}, \text{ego}, \dots)$ $K(K(F_0, \text{john}), \text{ego})$ - and common belief: $K(\text{formula}, \mathcal{X})$ when $K(\text{formula}, \mathcal{X} \dots)$ for all \mathcal{X} . Among the logics for belief in the literature, (Gar, KaFu, Kon, Lev, Weh), let us glance at the knowledge logic in (Kon). His S_0 , $[S_i]p$ and $\langle S_i, F \rangle p$ can be reformulated as \mathcal{X} , $K(p, \mathcal{X})$ and

$$\text{Proof}(p, \Gamma, \mathcal{X}) \quad \text{"p follows from } \Gamma \text{ in the logic of } \mathcal{X} \text{"}$$

and his sound and complete axiomatisation can also be reformulated. In this logic one can express " \mathcal{X} only believes what he can prove from Γ " by:

$$K(p, \mathcal{X}) \Leftrightarrow \text{Proof}(p, \Gamma, \mathcal{X}).$$

Belief logics capture natural and powerful arguments, a specialist domain expert may well decide to use a belief logic provided by a software tool.

#3.6 Knowledge

In many expert domains facts may only be what particular individuals or everybody knows. The facts in the knowledge base of a specialist in such a domain should be of the form

$$K(\text{formula}, \mathcal{K}, \dots)$$

where \mathcal{K} is some kind of "knower". The logic for the domain specialist should allow arguments about knowledge, in particular it should allow for self-knowledge, embedded-knowledge, and common knowledge:

$K(F, \text{ego})$, $K(K(F, \text{john}), \text{ego})$, $K(F, \mathcal{K})$ - the same formulae as before and we do not have to agree on whether knowledge is the same as true belief or not. Among the many knowledge logics in the literature (HaMo, MaSh, Ros, Sta, Var) the simplest is "many-agents S4" with rules:

$$K(\text{tautology}, \mathcal{K})$$

$$F :- K(F, \mathcal{K})$$

$$K(K(F, \mathcal{K})) :- K(F, \mathcal{K})$$

$$K(F_2, \mathcal{K}) :- K(F_1, \mathcal{K}) \quad K(F_1 \Rightarrow F_2, \mathcal{K})$$

A specialist domain expert may well decide to use a knowledge logic provided by a software tool.

#4. Dynamic Logics, Actions and Planning

Specialist domain experts can almost always perform actions that change their internal knowledge base K_0 and logic L_0 . In our free blackboard system specialists respond to a !-message by performing an action, and some of them may perform a plan of actions when they are sent a message

!Execute(plan)

where "plan" is an expression in some planning language. Where do plans come from? There seem to be three possibilities

- from the human user
- from the specialist itself using K_0 and L_0
- from the knowledge base and logic of a planning specialist.

If plans are to be created by the specialist itself or a planning specialist, then a logic of plans and actions must be available. Within such a logic one has an assertion like:

If a specialist with knowledge base K_0 and logic L_0 performs "plan" then its knowledge base becomes K_1 and its logic becomes L_1 .

That is true in the "ideal world" of the planner. Such an assertion may or may not be true in the "real world"; when the specialist actually executes the plan it may be interrupted and disturbed by obstacles so that the changes to its knowledge base and logic are not those intended and planned. We will ignore this distinction between the ideal and the real world, when we discuss various planning logics in this chapter.

When a domain expert with knowledge base K_0 and logic L_0 tries to execute a plan, it may well change its knowledge base to K_1 and its logic to L_1 by

- executing a "production rule": if Condition then Action;
- choosing a PROLOG rule with impure features like assert and retract;
- executing and assignment

A domain expert usually makes such changes when it receives a !-message and may make them when it receives a ? - or ??-messages. One way of modelling this is to let a logic L_1 have predicates - H -heuristics like Execute - such that an attempt to prove an H -formula changes the knowledge base K_0 and/or the logic as a "side effect". If the attempt is successful - $H(\dots) \in M_0$ - the side effect may depend on the actual proof; if it is unsuccessful, there may still be a side effect. As the side effect may produce an inconsistent knowledge base and logic, specialist domain experts must be designed with care; there is a need for software tools that help in the design of specialist domain experts that react appropriately and safely to sequences of messages and other plans from humans and other experts in their environment.

#4.1 Situations

Many expert systems exploit the idea "actions are functions", introduced by McCarthy in 1957 (McC1) and used in the STRIPS planner soon after (FiN2). The idea is that the facts in a specialist domain should be of the form

$$K(\text{formula}, f_1(..(f_n(s)....))$$

where s is "situation" and $f_1..f_n$ denote "actions". The logic for the domain specialist should allow for arguments about actions, it should allow rules like

$$K(\text{On table}(\text{object}), \text{putdown}(s)):- K(\text{Inhand}(\text{object}), s).$$

Such logics are plagued by the "frame" problem, the apparent need for a plethora of rules about facts that are not changed by the performance of an action, such as "putdown". If this frame problem is not unbearable, a specialist domain expert may well decide to use a situation logic provided by some software tool.

Although time logics are sometimes used for planning - used as if they were situation logics - situations are not the same as time instants.

#4.2 Dynamic logic

Many expert systems exploit the idea "actions are procedures", particularly those written in conventional programming languages (see #2.1). The facts in a specialist domain expert can be of the forms

$$K(\text{formula}, \alpha, \dots)$$

where α denotes a procedure. The logic for such a domain specialist should allow for arguments about actions, it should allow rules like

$$K(F, \alpha_1; \alpha_2) :- K(K(F, \alpha_2), \alpha_1)$$

$$K(F, \alpha_1; \alpha_2) :- K(K(F, \alpha_2), p_1).$$

One can find a multitude of suitable rules in the enormous literature on dynamic logic and other "logics of programs" (ClKo, Ko, Man, Par, Prt). In much of this research on programming logics one distinguishes between

- $[\alpha] F$ for $K(F, \alpha)$ when "F always true after α "
- $\langle \alpha \rangle F$ for $K(F, \alpha)$ when "F may be true after α "

and this distinction is often appropriate for domain experts. A specialist domain expert may well use a programming logic provided by some software tool.

#4.3 Structured actions

Many expert systems exploit the idea "plans and actions are structured, they are built from more primitive plans and actions". The facts in a specialist domain expert can be of the form

$$K(\alpha_0, \text{combinator}(\alpha_1 \dots \alpha_n))$$

where action α_0 is built from more primitive actions $\alpha_1 \dots \alpha_n$. The logic for such a domain specialist should allow for arguments about combinations of actions, and some appropriate logics have been suggested (Haa, McDl, Moo).

A good source of appropriate logics is recent work on the semantics of programming languages. In the new action semantics (Mos) one can find a deep analysis of the many ways in which programs can be built up from primitive actions. In the literature on Petri nets, CCS and CSP one can find a deep analysis of "plans as concurrent sets of primitive actions". One can distinguish between sequential and concurrent domain experts - the primitive actions for a sequential domain expert assert or retract single facts in its knowledge base and logic, the primitive actions of a concurrent domain expert assert or retract sets of facts in its knowledge base and logic. As fig. 3 shows, plans for a sequential domain expert should be sequences of primitive actions, while plans for a concurrent domain expert should be partially ordered sets of primitive actions. The figure also shows the underlying Kripke model of an expert system, the possible actions on the family $\{ \langle K_i, L_i \rangle \}$ of its knowledge bases and logics. The semantics of most of the logics in section 3 and 4 can be given in terms of Kripke models.

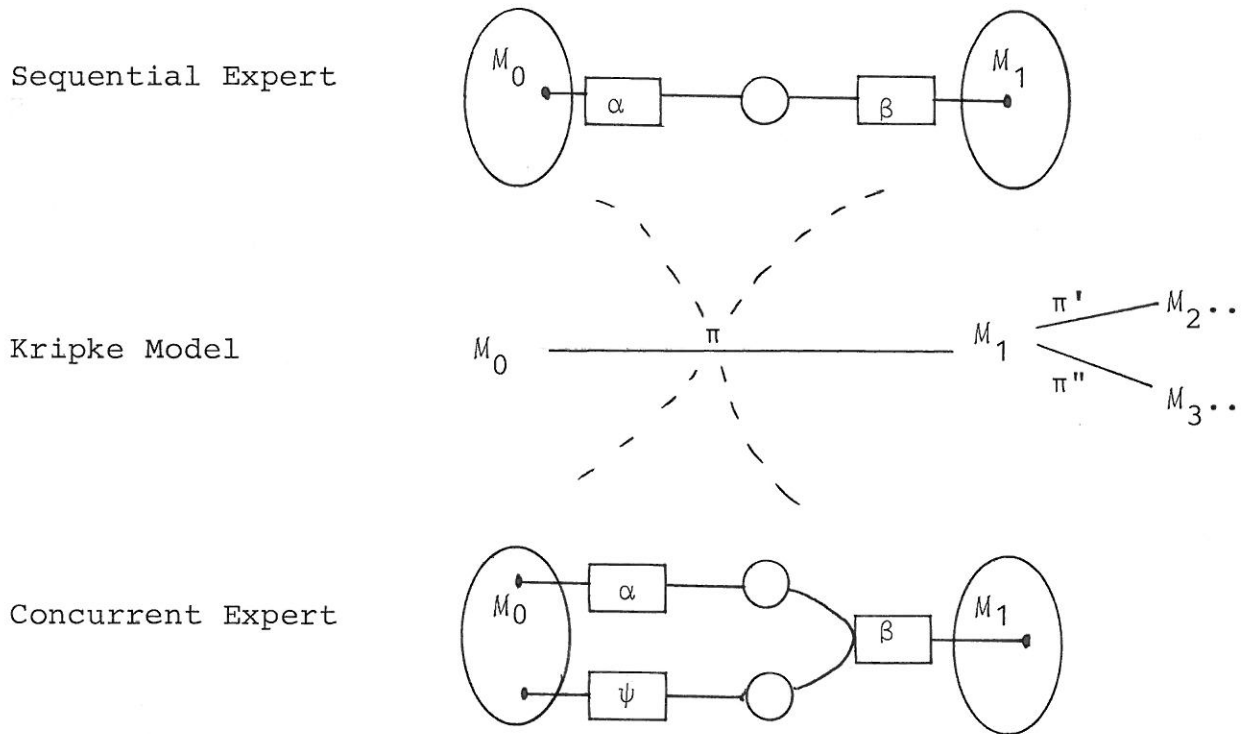


Fig. 3. Sequential and Concurrent Experts

There is a close connection between plans and heuristics. Every plan π determines a set of preconditions and a list of assertions and retractions of facts in a knowledge base and a logic. This can be captured by a heuristic H such that

$$H(A_1 \wedge \dots \wedge A_l \wedge \neg B_1 \wedge \dots \wedge \neg B_n) :- H(F_1) \dots H(F_m)$$

where $F_1 \dots F_m$ are the preconditions, $A_1 \dots A_l$ are the assertions, and $B_1 \dots B_n$ are the retractions. We have given the "PROLOG" form of the heuristic H , the production rule form would be

$$\underline{\text{if}} \ H(F_1) \ \underline{\text{and}} \ \dots \ H(F_m) \ \underline{\text{then}} \ \underline{\text{assert}} \ H(A_1) \ \underline{\text{and}} \ \dots \ \underline{\text{retract}} \ H(B_n).$$

One could say that heuristics are "frozen" plans, and the heuristics in #3 correspond to plans that do not change the knowledge base and logic of the expert.

The planning logics that are used in existing expert systems connect actions with time and other "state parameters" so they are not like the logic of "pure sets of actions" we have described briefly. Nevertheless software tools can deliver general planning logics to specialist domain experts that need to plan.

Example

Actions are primitive in (McD1). An expert \mathcal{E} doing an action is an event; $\text{Do}(\text{action}, \mathcal{E})$. In this planning logic one has actions, events, states, facts, tasks and chronicles; the logic of actions is embedded in a logic of time; in (Mor) a logic of actions is embedded in a logic of knowledge.

#5. Implementation

Our free blackboard organisation will be implemented in LOOPS on a Xerox 1108 (SB K). Most specialist domain experts will be LOOPS objects with "methods" for each !-, ?-, ??-message that the specialist can handle, Some specialist domain experts will be implemented on other machines connected to the Xerox blackboard; other machines will be used because

- they provide PROLOG, data bases, REVE rewriting, Petri nets and other convenient languages for writing efficient specialists
- they provide laser printers, speech recognition and generation, optical readers and other facilities for man-machine interaction
- many machines presumably give true parallelism and better performance.

By better performance we mean not only faster and more capacity, but also the improvement of the specialist by "learning from experience" and "reestablishing consistency" in its time free from handling blackboard messages.

What of our distinction between object level and metalevel facts and rules? We can collapse the levels by

- incorporating time, space, truth value, proof, believer, knower and other metalevel parameters as extra arguments to first order predicates
- formulating metalevel facts and rules as "rule set methods" for LOOPS objects.

Naturally we will provide software tools for each way of collapsing levels into LOOPS. There should be tools for some of the logics we have discussed, that collapses the logic into a LOOPS rule set L_{00} . A specialist domain expert that uses the logic will have L_{00} as its "unchangeable core"; during its life the specialist will have many knowledge bases K_i and logics L_i but L_{00} will always be part of L_i . There seem to be three possible locations for L_{00} in LOOPS:

- as another method in the specialist
- as a method inherited from a superclass of the specialist
- as a method in another "logic" specialist.

The choice between these possibilities is partly determined by the LOOPS "audit trail" facility, by the specialist's need to keep track of arguments so that it can learn from experience and/or explain its reasoning to a user.

Our implementation of the free blackboard organisation will have a number of disciplinary primitives for cancelling messages, interrupting specialists providing infinite answers to ??-messages, breaking deadlocks, and controlling anarchistic societies of specialists. These will be described in a later paper.

Epilog - from (Dea)

The most useful kind of expert system would an expert team consisting of the following: a skilled secretary who captures ideas, texts, and data items and arranges them neatly in labelled folders - a professor who coaches gently, when needed and asks whether something has been left out at times, but avoids imposing a straight jacket on the investigator - a librarian to look up information - a statistician to process data and help interpret the results - a writer to do the preliminary and final reports - an artist to produce the graphs and charts. A.G. Dean.

References

- (All 1) J.F. Allen; "Maintaining Knowledge about temporal intervals". Comm. ACM 26 (1983) 832-843.
- (All 2) J.F. Allen; "Towards a general theory of action and time". Art. Int. 23 (1984) 123-154.
- (AlHa) J.F. Allen, P. Hayes; "A common sense theory of time". Proc. 9 IJCAI (1985) 528-531.
- (VBen) J.F.A.K. van Benthem; "The logic of time". Reidel 1982.
- (Boo) G. Boolos; "The logic of provability". Am. Math. Mo. 91 (1984) 470-480.
- (BoKo) K.A. Bowen, R.A. Kowalski; "Amalgamating language and metalanguage in logic programming" in "Logic Programming". pp 153-172 ed. Tarnlund Academic 1982.
- (Bow) K.A. Bowen; "Metalevel programming and knowledge representation". New Gen. Com. 3 (1985) 359-383.
- (Bun) A. Bundy; "Discovery and reasoning in Mathematics". Proc. 9IJCAI (1985) 1221-1229.
- (Bur 1) J.P. Burgess; "Logic and time". J. Symb. Logic 44 (1979) 566-581.
- (Bur 2) J.P. Burgess; "Basic tense logic" in "Handbook of Philosophic Logic". ed. D. Gabbay, F. Guenther Reidel 1984.
- (ClKo) E. Clarke, D. Kozen (ed); "Logics of programs". Springer LNCS 164.
- (Coh) P.R. Cohen; "Heuristic reasoning about uncertainty: an AI approach". Pitman 1985.
- (Dea) A.G. Dean; "EPIAD". Byte (October 1985) 225-233.

- (Der) N. Dershowitz; "Synthetic programming".
Art. Int. 25 (1985) 323-373.
- (Doy) J. Doyle; "A truth maintenance system".
Art. Int. 12 (1979) 231-272.
- (EnGa) J.R. Ensor, J.D. Gabbe; "Transactional blackboards".
Proc. 9IJCAI (1985) 340-344.
- (EHLR) L.D. Erman, F. Hayes Roth, V. Lesser, D. Reddy;
"The hearsay II speech understanding system:
integrating knowledge to sleeve uncertainty".
ACM Comp. Sur. 12 (1980) 213-253.
- ((FaPr) H. Farrany, H. Prade; "A possibility theory - based
approach to default and exact reasoning".
Comp. Art. Int. 4 (1985) 125-136.
- (FiNi) R.E. Fikes, N.J. Nilsson; "STRIPS: a new approach
to the application of theorem proving to problem
solving". Art. Int. 2 (1971) 189-208.
- (FLM) R.E. Filman, J. Lamping, F.S. Montalvo;
"Metalanguage and metaplanning".
Proc. 8 INCAI (1983) 365-369.
- (Gar) P. Gardenfors; "Propositional logic based on the
dynamics of beliefs". J. Symb. Logic 50 (1985) 390-394.
- ((GoMe) J.P. Goguen, J. Mesequer, "EQLOG: equality, types and
generic modules for logic programming" in "Functional
and logical programming". ed. De Groot, Lindstrøm,
Prentice Hall 1985.
- (Goo) J.W. Goodwin; "A process theory of nonmonotonic in-
ference". Proc. 9 IJCAI (1985).
- (Haa) A. Haas; "Possible events, actual events and robots".
Comp. Int. 1 (1985) 59-70.
- (HaFa) J. Halpern, R. Fagin "Belief, awareness and limited
reasoning; preliminary report".
Proc. 9 IJCAI (1985) 491-501.

- (Hamo) J. Halpern, Y. Mosses; "A guide to modal logics of knowledge and belief: preliminary draft".
Proc. 9 IJCAI (1985) 480-490.
- (Hay) B. Hayes Roth; "A blackboard architecture for control".
Art. Int. 26 (1985) 251-321.
- (Hum) I.L. Humberstone; "Interval semantics for tense logic: some remarks". J. Phil. Logic 8 (1979) 171-196.
- (Jan) L.E. Janlert; "Studies in Knowledge representation".
Thesis (1985) Umeå.
- (KaGa) K. Kahn, G.A. Gorry; "Mechanising temporal knowledge".
Art. Int. 9 (1977) 87-108.
- (Kan) E.Yo. Kandrashina; "Representation of temporal knowledge".
Proc. 8 IJCAI (1983) 346-348.
- (Kon) K. Konolige; "Belief and incompleteness" in "Formal theories of the common sense world".
ed. Hobbs, Moore 1985 Ablex.
- (Koz) D. Kozen (ed); "Logics of programs". Springer LNCS 131.
- (LeEr) V.R. Lesser, L.O. Erman; "A retrospective view of the HEARSAY II architecture". Proc. 5 IJCAI (1977) 790-800.
- (Lev) H. Levesque; "A logic of implicit and explicit belief".
Proc. AAAI-84 pp 198-202.
- (LHBG) A. Lyall, P. Hammond, D. Brough, D. Glover;
"BIOLOG - a DNA sequence analysis system in PROLOG".
Nucleic Acid Res. 12 (1984).
- (MaBi) J. Malik, T.O. Binford; "Reasoning in time and space".
Proc. 8 IJCAI (1983) 343-345.
- (Man) Z. Manna; "Logics of programs". Proc. IFIP (1980) 41-52.
- (MaSh) J.P. Martins, S.C. Shapiro; "Reasoning in multiple belief spaces". Proc. 8 IJCAI (1983) 370-373.

- (May 1) B.H. Mayoh; "Galleries and institutions".
DAIMI PB 191 (1985) Århus.
- (May 2) B.H. Mayoh; "The connection between constructive
mathematics and computer programming".
Comp. Art. Int. 4 (1985) 203-209.
- (McC 1) J. McCarthy; "Situations, actions and causal laws".
AI memo 1 (1957) Stanford Univ.
- (McC 2) J. McCarthy; "Circumscription - a form of non-
monotonic reasoning". Art. Int. 13 (1980) 27-39.
- (McD 1) D. McDermott; "A temporal logic for reasoning about
processes and plans". Coyn. Sci 6 (1982) 101-155.
- (McDD) D. McDermott, J. Doyle; "Non-monotonic logic 1".
Art. Int. 13 (1980) 41-72.
- (McD 2) D. McDermott; "Non-monotonic logic 2".
J.A.C.M. 29 (1982) 33-57.
- (Moo) R.C. Moore; "A formal theory of knowledge and action"
in "Formal theories of the common sense world".
ed. Hobbs, Moore 1985 Ab lex.
- (Mos) P.D. Mosses; "Abstract semantic algebras" in
"Formal descriptions of programming concepts".
ed. D. Bjørner N. Holland 1983.
- (NiAi) H. Penny Nii, N. Aiello; "AGE (attempt to generalize):
an Knowledge based program for building knowledge and
programs".
Proc. 6 IJCAI (1979) 645-655.
- (OrPa) E. Orłowska, Z. Pawlak; "Logical foundations of
knowledge representation".
Fuzzy Sets & Sys. 1 (1983) 199-227.
- (Par) R. Parikh (ed); "Logics of programs".
Springer LNCS 193.

- (Prd) H. Prade; "A computational approach to approximate and plausible reasoning with applications to expert systems". Trans. I.E.E.E. PAMI 7 (1983) 260-283.
- (Prt) V.R. Pratt; "Process logic". Proc. 6 POPL (1979) 93-100.
- (ReSi) J. Reif, A.P. Sistla; "A multiprocess network logic with temporal and spatial modalities". JCCS 30 (1985) 41-53.
- (Rei) R. Reiter; "A logic for default reasoning". Art. Int. 13 (1980) 81-132.
- (ReUr) N. Rescher & A. Urguhart; "Temporal logic". Springer 1971.
- (Ros) S.J. Rosenschein; "Formal theories of knowledge in AI and robotics". New Gen. Comp. 3 (1985) 345-357.
- (San) E. Sandewall; "A functional approach to non-monotonic logic". Proc. 9 IJCAI (1985) 100-106.
- (Sho) Y. Shoham; "Ten requirements for a theory of change". New Gen. Comp. 3 (1985) 467-477.
- (Sta) W.R. Stark; "A logic of knowledge". Z. Math. Logic 27 (1981) 373-374.
- (Ste) M.J. Stefik, D.G. Bobrow, K.M. Kahn; "Integrating access-oriented programming in a multiparadigm environment". IEEE Software 3 (1986) 10-18.
- (StTh) R. Steinmetz, S. Thiessen; "Integration of Petri nets into a tool for consistency checking of expert systems with rule based knowledge". Proc. 6 Eur. W. App. Th. Petri Nets Espoo (1985) 35-52.

- (Var) M.Y. Vardi; "A model theoretic analysis of monotonic knowledge". Proc. 9 IJCAI (1985) 502-510.
- (Vil) M.B. Vilain; "A system for reasoning about time". Proc. AAAI (1982) 197-201.
- (Weh) R.W. Wehrauch; "Prolegomena to a theory of mechanized formal reasoning". Art. Int. 13 (1980) 133-170.
- (Zad) L.A. Zadeh; "PRUF - a meaning representation language for natural language". Int. J. Man-Machine St 10 (1978) 395-460.