IMPROVING THE EXPRESSIVENESS OF MANY SORTED LOGIC

Anthony G Cohn

Department of Computer Science
University of Warwick
Coventry CV4 7AL
England

Abstract

Many sorted logics can allow an increase in deductive efficiency by eliminating useless branches of the search space, but are usually formulated so that their expressive power is severely limited.

The many sorted logic described here is unusual in that the quantifiers are unsorted; the restriction on the range of a quantified variable derives from the argument positions of the function and predicate symbols that it occupies; associated with every non-logical symbol is a *sorting function* which describes how its sort varies with the sorts of its inputs; polymorphic functions and predicates are thus easily expressible and statements usually requiring several assertions may be compactly expressed by a single assertion.

The sort structure may be an arbitrary lattice. Increased expressiveness is obtained by allowing the sort of a term to be a more general sort than the sort of the argument position it occupies. Furthermore, by allowing three boolean sorts (representing 'true', 'false' and 'either true or false'), it is sometimes possible to detect that a formula is contradictory or tautologous without resort to general inference rules.

Inference rules for a resolution based system are discussed; these can be proved to be both sound and complete.

1. Introduction

Much research has been directed towards ways of cutting down the search space of mechanised inference systems. One approach is to divide the individuals in the intended interpretation into different *sorts* and then specify the sorts of the arguments of all the non-logical symbols in the language and the sorts of the results of function symbols; such a logic is known as a *many sorted logic(msl)*. Inference rules can be devised for such a logic so that many inferences which are obviously 'pointless' (to the human observer) can easily be detected to be such by the system because functions or predicates are being applied to arguments of inappropriate sorts. Sortal information can thus be viewed as a form of meta-knowledge.

Msls provide an simple syntactic way of specifying semantic information. Several mechanised msls have been proposed or built: eg (Reiter, 1981), (McSkimin, 1977), (Weyhrauch, 1978) and (Champeaux, 1978). Sorts in a logic are rather akin to types in conventional programming languages and problems often found in strongly typed programming languages may also occur in msls. In particular the typing/sorting mechanism often reduces the expressive power of the language: it is not long before a Pascal programmer becomes frustrated by the impossibility of writing general purpose library procedures because procedures may not be polymorphic.

In this paper we report on a msl which does not arbitrarily restrict the expressive power of the language and in which it is possible to specify detailed sortal information about the non-logical symbols which can then be used by the inference rules to reduce the search space. Space constraints only allow a discursive discussion of the logic. Full details including soundness and completeness proofs can be found in (Cohn, 1983).

This work has been supported in part by the SERC.

2. Preliminaries

We assume the reader is conversant with the First Order Predicate Calculus and resolution systems.

We use upper-case ***BOLD ITALIC*** letters (possibly with numeric suffices and/or primes) as meta-variables denoting expressions in the object language. We use bold Roman and Greek letters for all other meta-variables.

The alphabet of the language is the union of the following sets: **P**: a non-empty set of *predicate symbols* (we use strings composed of entirely of upper-case Roman letters), **F**: a non-empty set of *function symbols* (we use strings composed entirely of lower-case Roman letters or numerals), **V**: a non-empty set of *variables* (we use lower-case *italic* letters), $\{\vee,\wedge,\neg,\rightarrow,\leftrightarrow,\ldots\}$: a set of *boolean connectives*, $\{\bigwedge,\bigvee\}$: two quantifiers: the universal quantifier $\bigwedge$ and the existential quantifier $\bigvee$, $\{[,],',','\}$: three punctuation symbols.

Terms and formulae are formed from the alphabet in the usual way. Terms are either *variables* or *combinations*. Formulae are *atoms*, *literals*, *boolean combinations* or *quantifications*. Disjunctions of literals are called *clauses* and are usually represented simply as the set of their constituent literals.

3. The Sort Structure

Various sort structures occur naturally: disjoint sorts, trees and lattices. We choose to define the sort domain **S** as the most general such structure, a complete boolean lattice. An interpretation must interpret the top ($\top_{\mathbf{S}}$) and bottom ($\bot_{\mathbf{S}}$) elements of **S** as the universe of discourse (**U**) and the empty set respectively. The partial ordering on **S** is called $\sqsupseteq_{\mathbf{S}}$. We also need the lattice operators $\sqcup_{\mathbf{S}}$, $\sqcap_{\mathbf{S}}$ and $\setminus_{\mathbf{S}}$. We usually omit the subscripts on these symbols. It is useful to distinguish those elements in **S** immediately above $\bot$; these *disjoint* sorts (their interpretations are non overlapping sets) are denoted by $\mathbf{S}_1$*.

*In earlier presentations of this work the sense of the lattice was inverted, so that the interpretations of $\top$ and $\bot$ were reversed. The earlier convention followed the Scott-Strachey tradition, but the present convention is

Of course it is not necessary to actually name all the sorts in the sort lattice. It would be ridiculous to have to do so for one would then be forced to think up names for many sorts which might never actually occur in any assertion or during inference. Thus we can distinguish between *eponymous* and *anonymous* sorts. Every anonymous sort should be expressible (using $\sqcup$, $\sqcap$ and $\setminus$) purely in terms of eponymous sorts. However, since it is easy for an inference engine to invent names for all the anonymous sorts for internal use, we shall henceforth assume that all sorts are eponymous.

It will be seen later that we require a unary predicate symbol for each eponymous sort, so it will be convenient to use this predicate symbol as the sort's name. Literals formed from these symbols are called *characteristic* literals.

4. Sorting Functions

Following (Hayes, 1971) we describe the sortal behaviour of a function symbol α by a *sorting function* $\ddot{\alpha}$ of the same arity which maps $\mathbf{S}$ to $\mathbf{S}$. This allows the specification of *polymorphic* function symbols. The domain equation for sorting functions is $\mathbf{S}^n \rightarrow \mathbf{S}$. For technical reasons the crossproduct operation used to form the domain $\mathbf{S}^n$ is in fact not the usual pointwise operation; also, sorting functions may be quite reasonably required to be both *strict* and *continuous;* for details see (Cohn, 1983).

Sorting functions can also be used to describe the sortal behaviour of predicates, thus allowing polymorphic predicates. The question arises as to what the sort of an atom should be. One possibility is to invent a sort called BOOL $\in \mathbf{S}$ which is always interpreted as the set of the two truth values. This is the usual approach, but a better technique is to have a separate, boolean sort lattice, $\mathbf{B}$={EE,TT,FF,UU}. EE and UU are the $\bot$ and $\top$ elements of $\mathbf{B}$ respectively and TT and FF have fixed interpretations of {true} and {false} respectively. It is obvious that since $\mathbf{B}$ is a complete boolean(!) lattice it is just a particular $\mathbf{S}$ and all results for $\mathbf{S}$ apply to $\mathbf{B}$ as well. We can now give the functionality of sorting functions for predicate symbols; if $\alpha \in \mathbf{P}$ then $\ddot{\alpha}$: $\mathbf{S}^n \rightarrow \mathbf{B}$.

It is important to point out that we are *not* now dealing with a four or even three valued logic. The logic is still two valued; we interpret function symbols as partial functions but predicate symbols are interpreted as relations rather than partial predicates so in any interpretation a well-sorted formula (ie one whose sort is not EE) denotes either true or false; even those formulae whose sort is UU still denote one of the two truth values. (Our definition of satisfiability also ensures that all formulae sorted as TT or FF always denote true or false respectivcly). The four different boolean sorts exist purely for the benefit of the deductive machinery; formulae sorted as FF or TT can immediately be deduced to be contradictions or tautologies, as appropriate; those which are UU require inference in the usual way to determine a truth value. A formula whose sort is EE is ill-sorted; the deductive machinery will refuse to perform inferences with it and under no interpretation does it denote a truth value; it is as meaningless as if it were syntactically ill-formed.

5. Well-Sorted Formulae

Intuitively, a formula is well-sorted iff the sorts of all the sub-expressions 'match' the sorts required by their respective argument positions in a consistent manner. In most formulations of msl the sort $\tau 1$ of a term only matches the sort $\tau 2$ of its argument position if $\tau 1 \sqsubseteq \tau 2$. However the logic has more expressive power** if the match only fails when $\tau 1 \sqcap \tau 2 = \bot$. Eg, let $g \in \mathbf{F}$ have a sorting function such that

$$\ddot{g}(\langle MAN \rangle) = \bot \quad \& \quad \ddot{g}(\langle WOMAN \rangle) = MAN$$

where $\mathbf{S}_1$ = {MAN,WOMAN} and let c be a constant symbol of sort $\top$. We allow g[c] to be well-sorted even though if c is interpreted as a man then g[c] fails to denote.

Some machinery is required in order to be able to assign a sort to variables since variables do not have sorting functions but clearly the sort of an expression containing variables cannot, in general, be determined without knowing the sort of all constituent terms. The usual technique is to have separate quantifiers for every sort, for the sort of a variable could then be determined from the sort of its governing quantifier. However this would reduce the value of allowing polymorphic sorting functions since an instance of a variable would then have a unique sort associated with it. Eg suppose P is a rank two predicate symbol such that

$$\ddot{P}(\langle M,W \rangle) = \ddot{P}(\langle W,M \rangle) = EE$$
$$\ddot{P}(\langle M,M \rangle) = \ddot{P}(\langle W,W \rangle) = UU$$

where $\mathbf{S}_1$ = {M,W}. To express that P[x,y] is always true would seem to need two statements if we have sorted quantifiers:

$$\wedge_M x \wedge_M y P[x,y] \quad \text{and} \quad \wedge_W x \wedge_W y P[x,y]$$

It would be much more natural if we could just write: $\wedge x \wedge y P[x,y]$ and let x and y range over M and W as appropriate (ie y should always be of the same sort as x). This is the mechanism envisaged by (Hayes, 1971) and is certainly very convenient. The basic idea is that the sort of variable should be determined by the argument positions it occurs in. If it occurs as an argument to a polymorphic symbol then it may range over several sorts. In this case the sort of the entire expression may vary as a function of the sorts of such variables.

Defining the sorting functions for predicate symbols so that they are EE rather than FF for arguments for which they cannot be true reduces the need for explicit sortal preconditions on variables. Eg if $\mathbf{S}_1$ = {HUMAN,NATNUM} and we intend LE $\in \mathbf{P}$ to denote the 'less than' relation on the natural numbers, then by sorting LE so that

$$\ddot{LE}(\langle NATNUM,NATNUM \rangle) = UU$$
$$\ddot{LE}(\top \setminus \langle NATNUM,NATNUM \rangle) = EE$$

then we can write $\wedge x LE[0,x]$ rather than $\wedge x[NATNUM[x] \rightarrow LE[0,x]]$ as we would have to if LE were sorted so that $\ddot{LE}(\top \setminus \langle NATNUM,NATNUM \rangle) = FF$.

---

the one usually found in type hierarchies and powerset lattices.

** (Kowalski, 1971) attributes this idea and result to Gordon Plotkin.

To determine a unique sort for an expression with free variables we need to assign a sort to every free variable. Such an assignment can be viewed as a map from variables to sorts. We shall call such mappings from variables to sorts *sort environments*, since they can be regarded as a binding of sorts to variables. The domain $\mathbf{E}$ of sort environments can be given thus:*** $\mathbf{E} = \mathbf{V} \rightarrow \mathbf{S}$.

Given a sort environment we can give the sort of an expression even if it contains free variables. A complete description of the sortal behaviour of an expression can therefore be represented by a function from sort environments to sorts. The domain $\mathbf{A}$ of such functions is given by the equation: $\mathbf{A} = \mathbf{E} \rightarrow \mathbf{S}$. Obviously when the expression in question is a formula rather than a term, the domain equation for $\mathbf{A}$ particularises to $\mathbf{A} = \mathbf{E} \rightarrow \mathbf{B}$. Since a possible and natural representation of an element in $\mathbf{A}$ is an array, we refer to elements of $\mathbf{A}$ as *sort arrays (SAs)*.

In (Cohn, 1983) an algorithm is given to compute a SA $\tilde{\boldsymbol{B}}$ for any formula or term $\boldsymbol{B}$. Given $\xi \in \mathbf{E}$ then $\tilde{\boldsymbol{B}}(\xi)$ gives the sort of $\boldsymbol{B}$ in the environment ξ. If $\tilde{\boldsymbol{B}}(\xi)=\bot$ then $\boldsymbol{B}$ is ill-sorted in ξ. If $\forall(\xi \in \mathbf{E})\ \tilde{\boldsymbol{B}}(\xi)=\bot$ (ie if $\tilde{\boldsymbol{B}}=\bot_{\mathbf{A}}$) then $\boldsymbol{B}$ is ill-sorted. Conversely if $\forall(\xi \in \mathbf{E})\ \tilde{\boldsymbol{B}}(\xi)=\top$ (ie if $\tilde{\boldsymbol{B}}=\top_{\mathbf{A}}$) then $\tilde{\boldsymbol{B}}$ gives us no information about the precise sort of $\boldsymbol{B}$; we can regarded an unsorted logic as sorting all formulae and all combinations with $\top_{\mathbf{A}}$.

If $\boldsymbol{B}$ is a formula with a SA $\tilde{\boldsymbol{B}}$ then we can usefully distinguish the following two subsets of $\mathbf{A}$.

i) $\mathbf{A}_{\mathrm{TT}} = \{\tilde{\boldsymbol{B}} : \tilde{\boldsymbol{B}} \in \mathbf{A} \ \&\ \mathrm{rng}(\tilde{\boldsymbol{B}})=\{\mathrm{TT}\}\}$

ii) $\mathbf{A}_{\mathrm{FF}} = \{\tilde{\boldsymbol{B}} : \tilde{\boldsymbol{B}} \in \mathbf{A} \ \&\ \mathrm{rng}(\tilde{\boldsymbol{B}})=\{\mathrm{FF}\}\}$

where $\mathrm{rng}(\tilde{\boldsymbol{B}}) = \{\tilde{\boldsymbol{B}}(\xi) : \xi \in \mathbf{E} \ \&\ \tilde{\boldsymbol{B}}(\xi)\neq\bot\}$

If $\tilde{\boldsymbol{B}} \in \mathbf{A}_{\mathrm{TT}}$ then $\boldsymbol{B}$ has sort TT for all environments in which it makes sense and our definition of satisfiability ensures that $\boldsymbol{B}$ is true in all interpretations. If $\tilde{\boldsymbol{B}} \in \mathbf{A}_{\mathrm{FF}}$ then $\boldsymbol{B}$ has sort FF for all environments in which it makes sense and our definition of satisfiability ensures that $\boldsymbol{B}$ is false in all interpretations. If a well-sorted formula $\boldsymbol{B}$ has a SA $\tilde{\boldsymbol{B}} \notin (\mathbf{A}_{\mathrm{TT}} \cup \mathbf{A}_{\mathrm{FF}})$ then clearly $\mathrm{UU} \in \mathrm{rng}(\tilde{\boldsymbol{B}})$, so there are environments in which the truth of $\boldsymbol{B}$ cannot be inferred solely from sortal information; $\boldsymbol{B}$ might still be tautologous or contradictory but it requires the application of more general inference rules to detect this.

In (Cohn, 1983) the correctness of the SA algorithm is proved by showing a certain correspondence between an expression $\boldsymbol{B}$ and its SA $\tilde{\boldsymbol{B}}$. Essentially, the correspondence shown is that if $\tilde{\boldsymbol{B}}$ tells us that $\boldsymbol{B}$ is of sort τ then in any interpretation the denotation of $\boldsymbol{B}$ must be a member of the set denoted by τ in that interpretation. If $\boldsymbol{B}$ is a formula then this simply asserts that if $\boldsymbol{B}$ is of sort TT then it is valid and if it is of sort FF then it is unsatisfiable. Proving this correspondence thus amounts to a soundness theorem for SAs. If $\boldsymbol{B}$ is a combination or atom then we can also prove the converse, that if $\tilde{\boldsymbol{B}}$ informs us that $\boldsymbol{B}$ is of sort τ then for all $\tau' \sqsubseteq \tau$ s.t. $\tau'\neq\bot$ there is an interpretation in which $\boldsymbol{B}$ denotes an object in the set denoted by τ'. This means that $\tau = \mathrm{UU}$ only when necessary (ie when $\boldsymbol{B}$ is neither a tautology nor a falsehood). So if $\boldsymbol{B}$ is a tautology or a falsehood then it has sort TT or FF as appropriate. Thus we also have a form of completeness result for SAs.

It may also be noted that the SA algorithm can be used to detect integrity violations in a first order data base query language.

***For technical reasons, in (Cohn, 1983) the domain equation is actually somewhat different.

6. Many Sorted Inference Rules

A complete set of inference rules is given in (Cohn, 1983) which are shown to be sound and complete. Here we only have space to sketch what a set of resolution based inference rules for this msl might look like.

The transformation to clausal form is almost identical to the unsorted case except for the need to define the sorting functions for the skolem symbols. This is easily done using the SAs for the formulae in question.

6.1. Taking Instances.

We choose to view resolution as two rules: applying a substitution and resolution. The notion of an instance of a clause obtained by applying a substitution to a clause is different to the unsorted notion for several reasons. Firstly it may be that applying a substitution may yield an ill-sorted clause. Secondly, in some cases characteristic literals have to be added to clause. Eg let P be a predicate symbol sorted so that

$$\ddot{P}(<\tau>) = \mathrm{UU} \quad \& \quad \ddot{P}(<\top\backslash\tau>) = \mathrm{EE}$$

and let c be a constant of sort $\top$. Now both $\{P[x]\}$ and $\{P[c]\}$ are well-sorted, but it is not the case that $\{P[x]\} \models \{P[c]\}$ since any model σ of $\{P[x]\}$ where $\sigma(c) \notin \sigma(\tau)$ falsifies $\{P[c]\}$. Clearly it is only the case that $\{P[x]\} \models \{P[C]\}$ when the sort of C is $\sqsubseteq \tau$. However since P[c] is well-sorted we should like to be able to instantiate x to c. We can do this provided we broaden our notion of instance so that the instance of $\{P[x]\}$ obtained by substituting c for x is $\{\neg\tau[c], P[c]\}$. We shall call the characteristic literal $\neg\tau[c]$ a *prosthetic* literal since it has to be 'grafted' onto the clause $\{P[c]\}$ in order to preserve soundness.

It is also possible that applying a substitution may yield several clauses as an instance. Eg, let the rank two predicate symbol SPOUSE be sorted so that

$$\ddot{\mathrm{SPOUSE}}(<\mathrm{M},\mathrm{M}>) = \ddot{\mathrm{SPOUSE}}(<\mathrm{W},\mathrm{W}>) = \mathrm{EE}$$

$$\ddot{\mathrm{SPOUSE}}(<\mathrm{M},\mathrm{W}>) = \ddot{\mathrm{SPOUSE}}(<\mathrm{W},\mathrm{M}>) = \mathrm{UU}$$

where $\mathbf{S}_1 = \{\mathrm{M},\mathrm{W}\}$. Consider the (very polygamous!) clause $\{\mathrm{SPOUSE}[x,y]\}$. If we substitute c for x then there are in fact two instances****:

i) $\{\neg\mathrm{M}[c], \mathrm{SPOUSE}[c,y]\}$ ii) $\{\neg\mathrm{W}[c], \mathrm{SPOUSE}[c,y]\}$

We should not find it entirely surprising that applying a substitution may give rise to multiple instances for certain polymorphic formulae. As long as variables remain uninstantiated then a formula may be polymorphic; however a ground expression has a unique sort and thus can not be polymorphic. SAs can only represent sortal relationships between variables; as combinations are substituted for variables these relationships must be made explicit in the formula by prosthesis and multiple instancing. Although the

****The inference is in fact only sound if a mechanism is provided for restricting the quantification of y to $\{\mathrm{W}\}$ in (i) and to $\{\mathrm{M}\}$ in (ii). Details are in (Cohn, 1983).

possibly explosive effect of multiple instances on the search space during a proof may appear worrying it is certainly not a product of the msl. In an unsorted logic the original formulae would have been much more more complex or more numerous. Eg the many sorted formula: $\wedge x \wedge y$SPOUSE$[x,y]$ would have to be represented in an unsorted logic by a formula such as

$$\wedge x \wedge y[[[M[x] \barwedge W[y]] \vee [W[x] \barwedge M[y]] \rightarrow SPOUSE[x,y]]$$

Thus the search space is complex from the start even with uninstantiated variables. In an unsorted clausal resolution system the above formula translates to two clauses each with three literals. This immediately gives a choice of clauses to resolve the literal ¬SPOUSE$[m,n]$ with. By contrast, in our msl there would be but a single unit clause. Provided terms of sort M or of sort W were substituted for x and y then no prosthesis or multiple instancing would take place and the search space would be much simpler than in the unsorted case.

Of course if terms of sort $\top$ are substituted for both x and y then two instances each with two prosthetic literals would be inferrable; these two clauses would be identical to those inferred in an unsorted logic. However this should come as no surprise since we have no useful sort information about the terms substituted for x and y and thus the sort mechanism cannot operate effectively.

If the sort of the term substituted for x is M and the sort of the term substituted for y is $\top$ then there are two instances but with only one prosthetic literal in each clause. Thus the sortal mechanism delays prosthesis and multiple instancing until actually needed. In an unsorted logic the multiple clauses and the literals expressing the sort restrictions are always unavoidable.

6.2. Many Sorted Resolution

The resolution rule given in (Cohn, 1983) corresponds to the general resolution rule given in (Robinson, 1979) and (Kowalski, 1971). It is convenient to split the resolution rule into two: a rule for resolving non-characteristic literals and a rule for resolving characteristic literals. Except for the complications already discussed caused by applying a substitution the first rule is identical to the usual unsorted rule.

If the structure of **S** were represented as a set of axioms in a theory then we would not need a separate rule of inference for characteristic literals. However this would not be in the spirit of a msl: the sort machinery should cope directly with sortal inferences. Moreover the number of axioms needed to represent **S** would be very large for even a quite moderate sized **S** which would increase the search space and clog the system. Therefore we define a special rule for resolving characteristic literals so that, for example, M$[C]$ can be resolved against W$[C]$, even though the predicate symbols differ. Unlike ordinary resolution a literal in the resolvent may directly descend from the literals resolved upon. Eg if we are resolving against $\tau 1[C]$ and $\tau 2[C]$ and $\tau 1 \sqcap \tau 2 = \tau 3 \neq \bot$ then the literal $\tau 3[C]$ occurs in the resolvent.

6.3. Sortcasting

Characteristic literal resolution can be viewed as an inference rule that replaces the axioms that define the structure of **S**. We also need a rule that can directly use the information contained in the sorting functions for the non-logical symbols; otherwise we would need a set of axioms to encapsulate this knowledge. Eg consider the sorting function for the predicate symbol MOTHER:

$$\dot{M}OTHER(<M>) = FF \quad \& \quad \dot{M}OTHER(<W>) = UU$$

where $\mathbf{S}_1 = \{M,W\}$. We could 'translate' this sorting function to a formula in the logic thus: $\wedge x$[MOTHER$[x] \rightarrow$W$[x]]$. We could then use this formula to infer W[c] from MOTHER[c] when c is of sort $\top$. However for reasons already stated we prefer not to encumber a theory with these axioms which just duplicate information already present in the sorting functions. Thus we need a special rule to perform inferences such as the above. This rule is called *sortcasting* and can be viewed as a resolving of a literal against the imaginary formula expressing the information content of the sorting function for the predicate symbol of the literal concerned. The rule comes in two forms depending on whether the literal is positive or negative. In general, the situation is complicated by the necessity of considering several combinations simultaneously. Eg if

$$\dot{SPOUSE}(<M,W>) = \dot{SPOUSE}(<W,M>) = UU$$

$$\dot{SPOUSE}(<M,M>) = \dot{SPOUSE}(<W,W>) = EE$$

where $\mathbf{S}_1 = \{M,W\}$ and c1 and c2 are constants of sort $\top$, then we can infer both [M[c1]$\barwedge$M[c2]] and [W[c1]$\barwedge$W[c2]] from {SPOUSE[c1,c2]} by sortcasting.

7. Final Remarks

Use of a msl is not only notationally convenient, as workers in knowledge representation have found out (eg see (Hayes, 1971), (Hayes, 1978), (McDermott, 1982), but also can considerably reduce the size of the search space. Implementation of the logic outlined in this paper is in hand. We believe that such a system might prove to be a useful and efficient tool.

8. Acknowledgements

I would like to thank Pat Hayes, Ray Turner and everybody with whom I have discussed this work.

9. References

Champeaux, D, "A Theorem Prover Dating a Semantic Network," in *Proc AISB/GI Conf on AI*, Hamburg (1978).

Cohn, A G, "Mechanising a Particularly Expressive Many Sorted Logic," PhD Thesis, University of Essex (1983).

Hayes, P J, "A Logic of Actions," in *Machine Intelligence 6*, Edinburgh University Press (1971).

Hayes, P J, *Ontology of liquids*, University of Essex (1978).

Kowalski, R and Hayes, P J, "Lecture Notes on Automatic Theorem Proving," DCL Memo 40, University of Edinburgh (1971).

McDermott, D, "A Temporal Logic for Reasoning About Processes and Plans," *Cognitive Science* **6**(1982).

McSkimin, J R and Minker, J, "The Use of a Semantic Network in a Deductive Question Answering System," in *Proc IJCAI 5*, Cambridge (1977).

Reiter, R, "On the Integrity of Typed First Order Data Bases," in *Advances in Data Base Theory, Volume 1*, ed. H Gallaire, J Minker, J M Nicolas, Plenum Press (1981).

Robinson, J A, *Logic: Form and Function*, Edinburgh University Press (1979).

Weyhrauch, R, *Lecture Notes for a Summer-School*, Istituto di Elaborazione dell'Informazione, Pisa (1978).