

MECHANIZING A PARTICULARLY EXPRESSIVE MANY SORTED LOGIC

Anthony Cohn
Department of Computer Science
University of Essex, U.K.

ABSTRACT

A particularly expressive many sorted logic is presented along with an outline of an implementation. The logic allows functions and predicates to be sorted polymorphically and the sortspace to be partially ordered. It is shown that truth values may sometimes be inferred solely on the basis of sort information.

1. INTRODUCTION

A problem arising in many AI systems is that of matching two structures. The simplest form of matching is a purely 'syntactic' comparison of nested patterns organised hierarchically, see [6]. Tips of the patterns are either constants (which match only if equal) or variables (which match any pattern in which they do not occur). We consider matching processes which utilise sort information; these are still relatively simple and syntactic (and thus cheap) but offer considerable selectivity, since the mismatch of sorts can prevent unifications which lead to useless inferences (contrast, e.g., KRL [1] where 'matching' covers arbitrarily complex inferencing). Moreover, it is sometimes possible to infer the truth value of an expression merely on the basis of the sorts of some of its subexpressions. E.g., if *c* is female then FATHER(*c*) is definitely false.

2. ASSIGNING SORTS TO EXPRESSIONS

Mechanized sorted logics are not new (e.g., [2], [8]), but typically their expressive power has been limited by the restrictions on how expressions may be sorted; in particular, variables must usually be quantified over a single sort thus precluding polymorphism (though [8] allows a very limited polymorphism).

Defining polymorphic functions is a common and useful style of programming. We want to use polymorphism in axiomatisations. Hayes [4] describes a logic allowing polymorphically sorted relations and functions. (In [7] Strachey distinguishes 'parametric polymorphism' in which, e.g., `maplist` has sort $(\alpha \rightarrow \beta) \rightarrow \text{list}(\beta)$, from

'ad hoc polymorphism' or overloading; our implementation only handles overloading). Rather than variables having sorts, each relation and function *h* has an associated sorting function *h'* which specifies which sorts are legal arguments and for each legal combination of argument sorts, the sort of the value of the function. [5] makes heavy use of this logic.

Thus the sort restrictions are specified once only for every non-logical symbol and the sorts a variable may have in any wff are determined by context. This is more convenient than specifying the sorts locally in every wff.

As a simple example consider the following naturally polymorphic relation

`spouse: (MANxWOMAN  $\rightarrow$   $\mathcal{U}$ )+(WOMANxMAN  $\rightarrow$   $\mathcal{U}$ )`
(the choice of $\mathcal{U}$ as the range is explained in sec. 4).

In a monomorphically sorted logic, we must define `spouse: HUMANxHUMAN  $\rightarrow$   $\mathcal{U}$` which fails to capture all the constraints of our earlier definition; alternatively, if we allow a partial ordering on sorts (see sec. 3) then it is possible to define `SPOUSE` in terms of `WIFE` and `HUSBAND` with tighter sort restrictions, at the expense of forcing a two way split in the search space, depending on whether we assume `WIFE(x,y)` or `HUSBAND(x,y)`. Our implementation allows the or-branching implicit in a polymorphic function to be maintained internally in a single derivation rather than forcing a split into several derivations, increasing the search-space branching rate.

However, contrary to Hayes' claim in [4], mechanizing such a sort structure is not trivial.

E.g., if `g:(S1xS2  $\rightarrow$  S1)+(S2xS1  $\rightarrow$  S2)` then `{P(g(x,y),g(y,x)),P(u,u)}` should fail to unify,

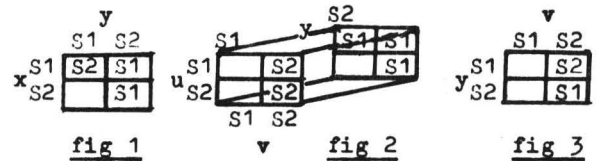
as should $\{Q(y, g(x, y), g(w, x)), Q(u, v, g(v, u))\}$, because the resulting terms $(g(x, x))$ in the former and $g(g(x, y), x)$ in the latter both violate g 's sort constraints. The difficulty occurs when merging two variables, because the checks cannot be kept local to the part of the proof tree being operated upon, as the scope of a variable is that of the whole proof tree. E.g., merging x/u would mean the erstwhile legal proof tree containing the terms $g(x, y)$, $g(y, z)$ and $g(z, u)$ would become illegal; locally each term looks correct; inconsistency is only detected when the three terms, which may be widely separated in the proof tree, are considered together.

We need to be able to check the sort restrictions globally. For every well formed expression $(wfe) b$, we can define a partial function $B: F \rightarrow S$ where $F = V \rightarrow S$, V is the set of variables and S the set of (for the moment) disjoint sorts, such that $B(f)$ is defined and gives the correct sort for the wfe for the combination of sorts on variables specified by f . We can regard $f \in F$ as specifying an environment in which every variable is bound to some sort. B then specifies which environments are permissible for b , and what the sort of b would be in that environment.

In our implementation, we choose to represent B as an n -dimensional square array called a sort array (SA), with as many rows and columns as there are sorts and $n = |V|$. Thus every position represents an $f \in F$. We define $\text{dom}B = \{f: B(f) \in S\}$. If sorts are being used to good effect then $|\text{dom}B|$ will be small and thus the array sparse. If $\text{dom}B = \emptyset$ then no combination of sorts of variables is legal: b is illformed.

We can compute the SA for a proof tree incrementally as the tree is built; the operations on SAs that we need to perform mirror the operations on the tree: substituting for variables and attaching new expressions to the proof. The primitive building blocks are the SAs representing the sorting functions of nonlogical symbols.

The following example illustrates two of the basic operations that we need to perform on SAs. Suppose the SA for term $g(x, y)$ is as in Fig. 1, then binding x to $g(u, v)$ gives the SA for $g(g(u, v), y)$ in Fig. 2. Binding a variable to a term with k new variables increases the dimensionality of the SA by $k-1$. Merging two variables decreases the dimensionality by one, by taking a diagonal slice through the array. So substituting u/y picks out the rows on the $y=u$ diagonal, giving the SA for $g(g(y, v), y)$ (see Fig. 3). Binding v to a constant c of sort $S1$ gives a vector with no defined entries; thus $g(g(y, c), y)$ is not a wfe .



3. NON-DISJOINT SORTS

A sort specifies a subset of objects in the universe. If we allow non-empty intersections then we naturally induce a partial ordering (i.e., set inclusion) on S . Matching the sorts of two terms when unifying is then set intersection.

Inclusion induces a partial ordering on F and $F \rightarrow S$ in the usual way, e.g.:

$\forall f_1, f_2 \in F \quad f_1 \supseteq f_2 \equiv \forall v \in V \quad f_1(v) \supseteq f_2(v)$

We require B in $F \rightarrow S$ to be monotonic:

$f_1 \supseteq f_2 \Rightarrow B(f_1) \supseteq B(f_2)$

and sensible:

$\forall d \subseteq \text{dom}B, d \neq \emptyset: \exists z = \bigcup d \Rightarrow \exists y = \bigcup \{B(f): f \in d\}$

With these (natural) restrictions, the implementation sketched above can be extended to the case of non-disjoint sorts. Considerable computational and space savings follow from the observation that one can often calculate some entries in a SA, given others. Thus only part of each SA - already sparse - needs to be explicitly stored in memory. Details of such normal SAs are in [3].

4. SORTING WFFS.

The value of a wff is true or false. Defining a single sort $\mathcal{U} = \{\text{true}, \text{false}\}$ leads to problems: we can only sort MOTHER and FATHER loosely as $(\text{HUMAN} \times \text{HUMAN} \rightarrow \mathcal{U})$ to make sense of $\forall x, y \text{ PARENT}(x, y) \equiv \text{MOTHER}(x, y) \vee \text{FATHER}(x, y)$ otherwise $x \in \text{MAN} \wedge y \in \text{WOMAN}$! We shall define the functionality of SAs for wffs as $F \rightarrow \text{BOOL}$ where $\text{BOOL} = \{\mathcal{T}, \mathcal{F}, \mathcal{U}\}$ and $\mathcal{F} = \{\text{false}\}$ and $\mathcal{T} = \{\text{true}\}$. We can induce a lower semi-lattice on BOOL as we did on S . We now sort MOTHER and FATHER as $(\text{MAN} \times \text{HUMAN} \rightarrow \mathcal{F}) + (\text{WOMAN} \times \text{HUMAN} \rightarrow \mathcal{U})$ and $(\text{MAN} \times \text{HUMAN} \rightarrow \mathcal{U}) + (\text{WOMAN} \times \text{HUMAN} \rightarrow \mathcal{F})$ respectively.

An advantage of this technique is that characteristic predicates need no special treatment by the inference machine; e.g., we can sort $\text{MAN}: (\text{MAN} \rightarrow \mathcal{T}) + (\mathcal{U} \setminus \text{MAN} \rightarrow \mathcal{F})$.

To demonstrate a second advantage let $r = \{B(f): f \in F\}$ where b is some wff; if $r = \{\mathcal{F}\}$ then b is a contradiction (e.g., $\text{MOTHER}(\text{John}, x)$) and if $r = \{\mathcal{T}\}$ then b is a tautology (e.g., $\neg \text{MOTHER}(\text{John}, x)$). Similarly if we are trying to prove $\forall x, y \text{ MOTHER}(x, y)$ and we resolve against the above assertion then we need only resolve away the subgoal $\text{PARENT}(x, y)$ since $\text{FATHER}(x, y)$

can be inferred false by the sort structure. This technique generalises the concept of insignificant literal in [2].

5. THE EFFECT OF SORTS ON THE LOGIC

We can only sketch this interaction: full details and proofs, including completeness and soundness results will be found in [3].

Every SA represents a set of axioms which would otherwise have to be explicitly asserted; e.g., if $b: (S1 \times S2 \rightarrow S3) + (S4 \times S5 \rightarrow S6)$

then the missing axioms are (assuming disjoint sorts): $\forall x, y S3(b(x, y)) \equiv S1(x) \wedge S2(y)$ and $\forall x, y S6(b(x, y)) \equiv S4(x) \wedge S5(y)$

We must take account of these implicit axioms if we are to prove (say) $S2(c)$, for a proof of $S3(b(c, y))$ would imply $S2(c)$. That a given SA B is correct may be shown by proving that the equivalent axioms follow from the sort lattice axioms, the primitive SAs for all the nonlogical symbols and from b itself.

The axioms which define the structure of the sort lattice are eliminated in two ways.

(1) Use of lattice theoretic operations (e.g. $\sqcup, \sqcap$) on SAs (see definitions below).

(2) By extending resolution in the following way. We must allow $Si(c)$ to match with any positive literal $Sj(t)$ (where $Sj \sqcup Sc \sqsubseteq Si$, t unifies with c, $Sc, Si, Sj \in S$ and $Sc(c)$). This includes the conventional case where $Sj = Si$. It must also match $Sj(t)$ where $Sj \sqcup Sc \sqsubseteq Si$ provided we also prove $\neg(Sj Si)(c)\sigma$, where σ is the substitution unifying t and c. Similar but dual techniques apply when trying to prove $\neg Si(c)$.

We now present a formal definition of how to compute the new SA D after substituting e for x in a proof tree b. We need only consider the modified SA B for which $\{B(f): f \in \text{dom}B\} = \{\mathcal{U}\}$, since for all other $f \in \text{dom}B$ we can already infer b's truthvalue. So clearly if $f \in \text{dom}D$ then $D(f) = \mathcal{U}$. There are two cases to consider

(1) e is a variable then

$\forall f \in \text{dom}B \exists z = f(x) \sqcup f(e) \Rightarrow$

$(\lambda v. \bar{v} = e \rightarrow z, \forall \bar{x} \rightarrow f(v)) \in \text{dom}D$

(2) e is not a variable

$\forall f \in \text{dom}B, g \in \text{dom}E f(x) \sqsubseteq E(g) \Rightarrow$

$(\lambda v. \bar{v} \bar{x} \rightarrow (f + g)(v)) \in \text{dom}D$

If necessary b and e must be standardised apart before computing D. Subsequently the erstwhile shared variables must be remerged, diagonalizing in each case.

(Both the above operations have to be modified slightly to propagate normality.) Definition (2) only allows a term to match a variable of a lesser sort, though we can remove this restriction by replacing the left hand side with ' $\exists z = f(x) \sqcup E(g)$ ', providing that we also prove that e is of the more restricted sort; the re-

sulting logic is more expressive (noticed first by Plotkin). E.g., suppose we wish to prove $\exists x P(x)$ where P is only defined on abelian groups. If we know that $R(g)$ and that $\forall x R(x) \Rightarrow P(x)$ (where R is defined on groups and g is some group) then we can complete the proof providing we also prove that g is abelian (clearly abelian-group $\sqsupset$ group). Polymorphism complicates the precise definition of this extension so we do not give it here.

6. CONCLUSION

The difficulty experienced in mechanizing a theory or language is usually proportional to its generality or expressiveness. We have proposed a particularly general many sorted logic: allowing polymorphic functions and a partially ordered sortspace; furthermore, the sort of an expression may sometimes be determined solely from sort information.

A pilot implementation which works for the basic theory is currently being extended. At present SAs have to be ground instances, i.e., sort meta-variables, allowing parametric polymorphism are not permitted. It is hoped to include this facility.

7. ACKNOWLEDGEMENTS

Phil Collier and Jon Cunningham made useful comments on a draft; I am particularly indebted to Pat Hayes for his help and guidance.

REFERENCES

- [1] Bobrow, D. and Winograd, T. "An overview of KRL", *Cognitive Science*, vol. 1, No. 1, (1977).
- [2] Champeaux, D. "Dating semantic networks", *Proc. AISB/GI, Hamburg*, 1978.
- [3] Cohn, A. "Mechanizing and implementing many sorted logic", Ph.D. thesis. In preparation.
- [4] Hayes, P. "A logic of actions", *Machine Intelligence* 6.
- [5] Hayes, P. "Naive Physics 1: Ontology of liquids", Essex University, 1978.
- [6] Robinson, J. "A machine oriented logic based on the resolution principle", *JACM* 22, pp.572-596.
- [7] Strachey, C. "Fundamental concepts in programming languages", Notes for the Int. summer school in computer programming, Copenhagen, 1967.
- [8] Weyhrauch, R. "Lectures notes for summer school on AI in Pisa", 1978.