An Improved Filter for literal Indexing in Resolution Systems

Lawrence J. Henschen

Northwestern University, Evanston. IL

Shamim A. Naqvi

Bell Laboratories, Naperville, IL

1. Introduction

The more successful resolution programs today owe their power in part to efficient retrieval mechanisms for returning literals that are potentially unifiahle with or subsumed by a given literal. For example, the efficient subsumption algorithms and the hack demodulation of [2] are based on one such scheme, the FPA lists. We propose a new technique that can be used to locate "compatible" literals for resolution, subsumption, etc. This new scheme has some advantages (and disadvantages) relative to FPA lists.

We make no distinction between atoms and non-variable terms. We call predicate, function and constant symbols rigid and variables flexible. We are therefore interested in fast retrieval and comparison of formulas that start with a rigid symbol.

2. Formula Outlines

Let T1 and T2 be two wffs. A necessary but not sufficient condition for T1 and T2 to unify is that T1 and T2 agree at every rigid-rigid position (i.e., corresponding positions in which both T1 and T2 contain rigid symbols). A necessary but not sufficient condition for T1 to subsume T2 is that in addition to the above, T2 contains a rigid symbol in every position where T1 contains a rigid position.

We make use of the above facts by grouping together wffs that have the same rigid symbols in the same positions. Further, we associate with each such group an expression called the outline. These outlines can be easily compared to test for the above conditions. Finally, such a comparison need be made only once for each pair of outlines. Compatible outlines can be linked, giving immediate access to all groups of wffs that potentially unify (subsume, are subsumed by) a given wff or outline group.

An outline consists of two byte strings, the symbol string and the indicator string, whose length depends on the depth of the atom. The first byte of each string contains information for the starting symbol of the wff, the next n bytes represent information for the starting symbols of the top level arguments of the wff, the next n**2 bytes represent information for the level 2 arguments in the wff, and so on. Unfortunately, in order that corresponding positions for nested subarguments line up in separate outlines, we must allow for n arguments for every symbol (even variables) where n is the maximum number of arguments for any symbol in the problem. (See below for a discussion of storage requirements.) The symbol string contains symbols for the rigid positions and zeros for the flexible or non-existent positions in the atom. The indicator string contains a byte of 1's in each rigid position and zeros in each flexible position. In the following examples, n=3 and we write each symbol byte with a letter or a zero, and each indicator byte with a single 1 or 0.

Example 1. P(g(x),a,f(a,b,y))

symbol string:	P g a f 0 0 0 0 0 0 a b 0
indicator string:	1 1 1 1 0 0 0 0 0 0 1 1 0

In this example, the 0 in the fifth position represents the variable x which is the first argument of the first argument of the atom. The next fives 0's are present because there are no corresponding arguments. This guarantees that the positions of the arguments of f in the third argument position of P will line up with the corresponding positions in another outline even though g and a do not have a full three arguments. The last zero is for the variable y.

Example 2. P(x,y,f(y,b,z))

symbol string:	P 0 0 f 0 0 0 0 0 0 0 b 0
indicator string:	1 0 0 1 0 0 0 0 0 0 0 1 0

Example 3. P(x,x,f(x,b,x))

Strings are the same as in Example 2.

Example 4. P(a,b,c)

symbol string:	P a b c
indicator string:	1 1 1 1

3. Comparison of Outlines

Let T1 and T2 have the outlines (S1,I1) and (S2,I2) respectively. Using standard string operations and padding shorter strings with zeros, let

I = NOT(I1 XOR I2)
R1 = S1 AND I
R2 = S2 AND I

We note the following:

1. I contains a 1 byte in exactly those positions that have rigid-rigid or flexible-flexible pairs.
2. R1 (R2) = S1 (S2) in rigid-rigid positions.
3. T1 and T2 unifiable implies that R1 = R2

4. T1 subsumes T2 implies the above condition and the string ((NOT(I2) AND I1) is identically 0, i.e., no rigid position in T1 is flexible in T2.

If T1 and T2 are the wffs of examples 1 and 2, then

```
      I = 1 0 0 1 0 0 0 0 0 0 0 1 0
R1=R2 = P 0 0 f 0 0 0 0 0 0 0 b 0
```

indicating potential unifiability. However, T1 cannot subsume T2 because of the variables x and y vs. g and a.

Any new wff whose outline already exists is simply added to that group. Otherwise, the new outline is added and the comparisons made and stored.

4. Notes

The first major difference between FPA and outlines is that with outlines one can get the appropriate list of wffs directly by retrieving the compatible outline groups. Briefly the FPA list n1.n2. ... nk.s contains all those formulas in which there occurs another formula starting with s in position n1....nk. If s is -1 then the term *is* a variable. For example, the atom in Example 1 would be on the P list, the 11-1 list, etc. Given a particular atom T, to find the atoms that may unify with (subsume, be subsumed by) T, it *is* necessary to take unions and intersections of appropriate FPA lists. Of course, in order to form new resolvents, one must find the clauses containing the literals in both schemes.

Several systems employ structure sharing. In the structure sharing scheme of Boyer-Moore [1], one would only need to store the outline string and then include for each wff in that outline group a means to tell which variables occurred in which zero positions. In the case of sharing as in the Overbeek et. al. program [3], one could use the outline group as an alternative to hashing terms. This would be less efficient in time, but would not be as wasteful of permanent memory.

We note that the expression I (-NOT(I1 XOR 12)) can also be used to determine those positions within the two terms that actually need to be tested for unification. Programs that can access subterms by position vectors can avoid the redundant checking of rigid-rigid pairs.

There is considerable wasted space in an outline string for argument positions that are non-existent. If the maximum number of arguments is big or the formulas are deep, the outlines can get very long. However, there *is* only one copy of each outline. In view of the possible computational advantages, it may not be out of line to use 30 to 40k bytes for these strings and the lists of compatible outlines. Indeed, because a given atom or term can occur on many FPA lists, it is not clear that outlines would require disproportionately more storage. In fact, the two schemes could be used together by cutting off the outlines at some level and using the FPA lists for terms deeper than the limit. The outlines would require very little storage and would provide a very good first filter; the deep FPA lists could be used as a secondary filter.

5. Acknowledgements

We with to thank Dr. Neil Haller and Mr. Robert Veroff for reading the original manuscript, and to Ms. Sherrie Brown for helping in the preparation of the final copy.

6.. References

[1] Boyer, R.S. and Moore, J.S. "The Sharing of Structure in Theorem Proving Programs", *Machine Intelligence* 7, Meltzer and Michie (eds.), Edinburgh Univ. Press, 1972, pp. 101-116.

[2] Overbeek, R. "An Implementation of Hyper-resolution", *Computer snd Ms thems tics with Applications,* Vol. 1, 1975, pp. 201-214.

(3] Overbeek, R. and Lusk, E. "Data Structures and Control Architectures for Implementation of Theorem Proving Programs", *Proc. of the 5th Conf. on Automated Deduction,* Springer-Verlag, 1980. pp. 232-249.