CONNECTION GRAPHS AND DECIDABLE THEORIES

Allan Ramsay

1987

Cognitive Studies Research Paper

Serial No. 079

Cognitive Studies Programme,
The University of Sussex,
School of Cognitive Science
Falmer, Brighton BN1 9QN

079

Connection graphs and decidable theories

Allan Ramsay
Cognitive Studies Programme
University of Sussex. Brighton BN1 9QN

Abstract
The expressiveness of fast order predicate calculus (FOPC) has led many people to use it as a representation language within computer systems. Intensive research over recent years has produced a number of efficient theorem proving techniques for FOPC, but they are all clearly subject to the limiting result that FOPC is in general semi-decidable. This paper presents a technique for investigating the decidability of particular theories of FOPC with respect to a particular theorem prover. Armed with this technique, we can frequently show that specific problems are decidable, in which case we can afford to investigate our queries for both proofs and disproofs.

1 Introduction

One of the most widely known meta-theorems about the predicate calculus (henceforth FOPC) is that it is semi-decidable (see for example (Boolos & Jeffrey 1980)). In other words, it is not possible to produce an algorithm to see whether a set of axioms entails either a target formula or its negation. This result has had three direct consequences for the use of FOPC in computer systems, as follow, (i) Since any complete theorem prover may simply fail to return when asked to prove a theorem for which no proof exists, all practical theorem provers have to made artificially incomplete, for instance by setting a resource limit. But once a theorem prover is known to be incomplete, it becomes very dangerous to interpret failure to prove a result as anything stronger than "unknown". GO The restriction of theorem proving techniques to make them incomplete must, by definition, mean that there will be valid inferences that will not be found. This leads to the unsatisfactory situation where although what you are writing appears to be in FOPC, the effective semantics differs since things which are valid conclusions in FOPC itself will fail to be derivable within the system. This can be clearly seen in PROLOG'S failure to prove the conclusion of Moore's well-known 'three blocks problem'[1]. This, in conjunction with the interpretation of 'negation as failure', means that the semantics of PROLOG are in fact quite radically different from the semantics of FOPC. (iii) The use of FOPC as a representation language for databases is constrained by the theoretical impossibility of checking consistency (which in turn follows from semi-decidability). It is most undesirable to allow inconsistent data to be entered into a database. This is particularly serious if you are using FOPC as the representation language and have a complete theorem prover, since in this case the presence of inconsistencies will enable the theorem prover to derive any conclusion whatsoever.

These constraints have had a knock-on effect, whereby people are pessimistic about using FOPC even when these particular problems are ignored. Practice shows, however, that theorem provers are hardly ever actually set problems which lead them into infinite searches for proofs of non-provable goals. The theoretical possibility of this happening seems to have swamped people's perception of the real situation. With decent modern theorem provers and ordinary problems, rather than pathological ones created merely as tests, infinite searches for non-existent proofs simply do not arise (Moore's three blocks problem is an example of an ordinary problem - it is only difficult if the theorem prover uses linear-input as its search strategy). Furthermore, it is often quite easy to see, even before you start trying to derive a proof, that whether or not you succeed in getting a proof, you are definitely not going to end up in an infinite search.

The aim of this paper is to restore confidence in the use of FOPC by showing how to extract easy proofs that a particular theorem proving technique (Kowalski's (1975) connection graph method) will definitely terminate for particular problems. The connection graph is a very difficult structure to prove general positive results about. Eisinger (1986) shows that in fact completeness itself is not attainable unless you are very careful about the heuristics you use to control the search, though since set-of-support and unit-preference are apparently both permissible this is not as pessimistic as result as it seems at first sight. A general method of testing whether a problem is decidable by a given theorem prover is in any case an impossibility, since such a method would solve the decision procedure for FOPC and thence for Turing machines in general. The algorithm we present here is a very simple conservative test. If it says the problem is decidable, it is. There will also be problems which are decidable which it cannot detect. It will become apparent that our algorithm could be extended to detect more cases, at the cost of making it more expensive.

The trade-off between putting effort into assessing whether a problem is likely to cause trouble or putting it into actually trying to find a solution must depend on the domain. All we claim here is that the existence of a cheap way of pre-checking queries should raise the general level of confidence in FOPC as a representation language.

2 Connection graphs

There is currently a debate over the best methodology for theorem proving for FOPC, with the front runners being Kowalski's (1975) connection graph (henceforth CG) and Bibel's (1982) connection method. These two are, despite their names, quite different approaches. It is still very unclear which is more efficient for FOPC, though there are indications that the connection method may be more easily adapted for more or less powerful systems such as modal logic or intuitionistic logic (Wallen & Wilson 1987). The connection graph, however, does have the advantage that the entire structure of the problem is easily accessible, so that it can be investigated by meta-level processes. It is this very accessibility of the graph that leads to the performance gains of this method, and it is also what makes the decidability algorithm feasible. It is not appropriate here to give a detailed description of the CG method or to argue for or against it on efficiency grounds, but we will need to give an overview, especially of the major structures involved, in order to explain how you can see whether a problem is going to be decidable before you even try to solve it.

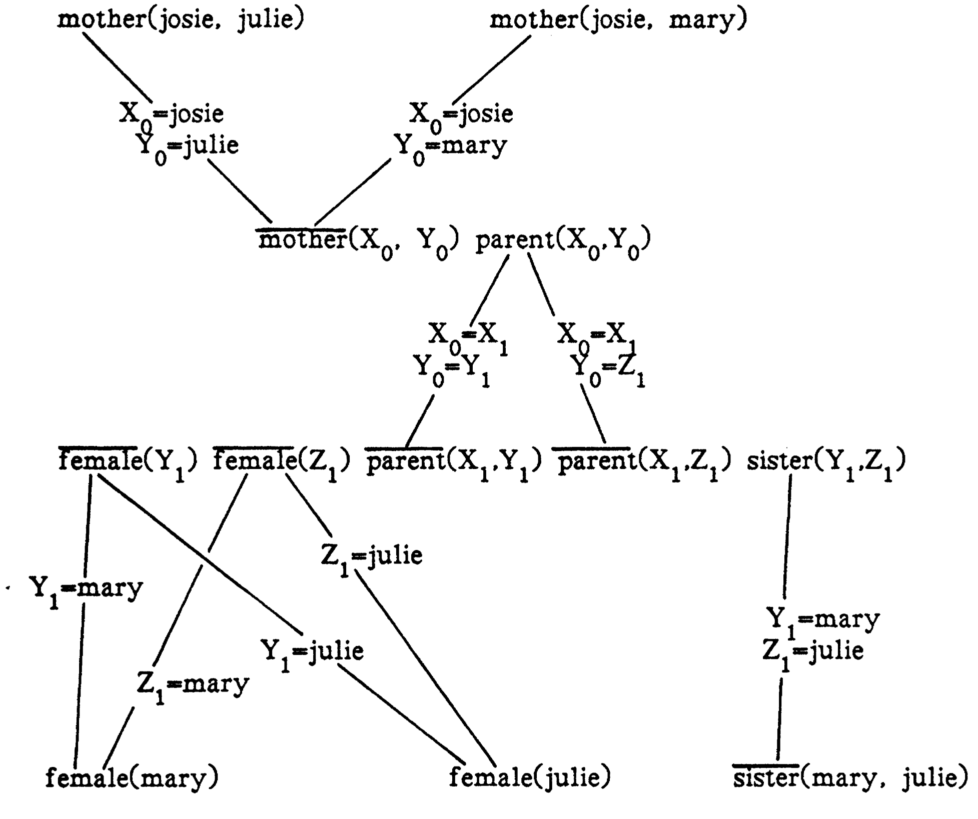
The CG is a resolution based approach to theorem proving. Most reasonable implementations of resolution index potentially resolving clauses via the contributing literals. This can be done at the point where the axioms and goal are converted to clausal form, so it costs very little. There is no extra process which has to be run over the clause set to create the index, since the literals become visible during conversion to clausal form. The CG is in essence just a particularly neat index, in which potentially unifiable literals are connected by links, which can in turn be labelled by the substitutions required to perform the unification. When resolutions are performed in the CG, the existing index is used to make sure that the resolvant is immediately linked to those clauses which were already indexed against the literals in the contributing clauses. To see this in concrete terms, consider the following very simple problem about whether two people are sisters:

Axioms:
$\forall(X,Y)(\text{mother}(X, Y) \rightarrow \text{parent}(X, Y))$
$\forall(X,Y,Z)(\text{parent}(X,Y) \,\&\, \text{parent}(X,Z) \,\&\, \text{female}(Y) \,\&\, \text{female}(Z) \rightarrow \text{sister}(Y,Z))$
mother(josie, julie)
mother(josie, mary)
female(mary)
female(julie)

Goal: sister(julie,mary)

Example 1: Sisterhood

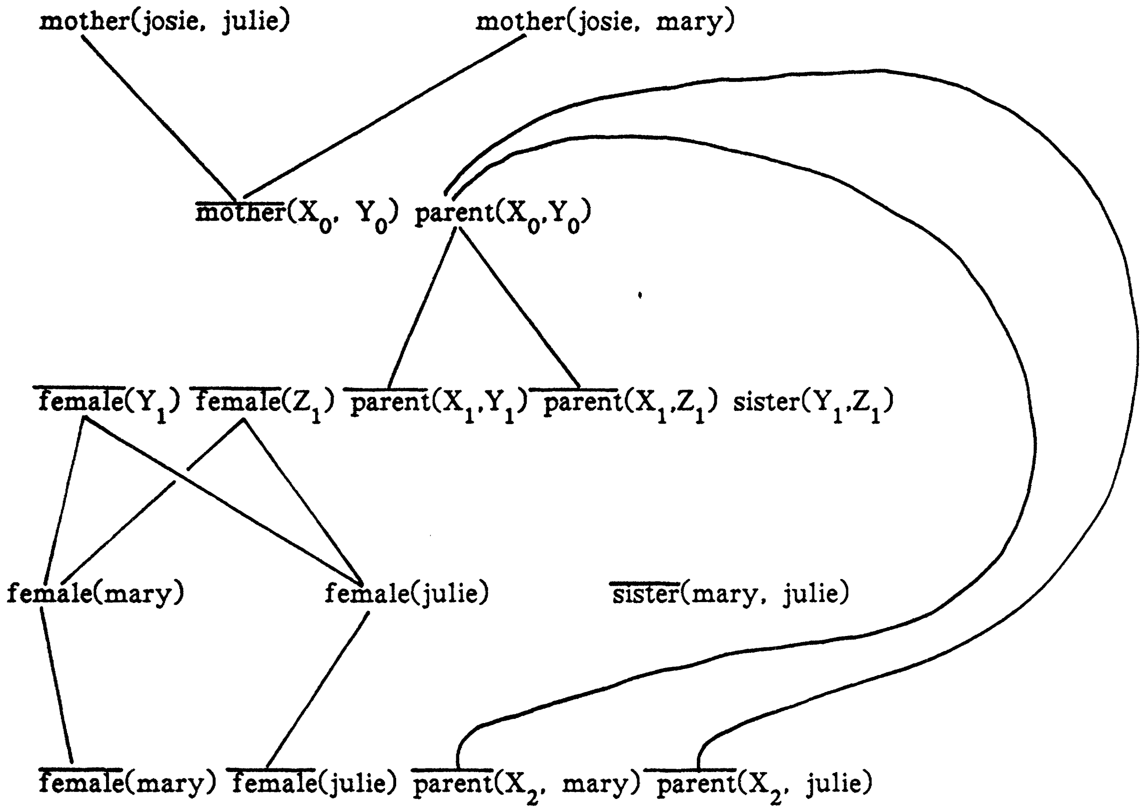
This converts to the following graph, where links between literals indicate that they were noted as being potentially unifiable. A line over a literal denotes that it is negated. The links in this graph have their unifiers displayed. In subsequent graphs, we omit the unifiers to keep the diagrams as simple as possible, but it should be remembered that the unifiers are always there.



mother(josie, julie)
mother(josie, mary)
X_0=josie Y_0=julie
X_0=josie Y_0=mary
$\overline{\text{mother}}(X_0, Y_0)$ parent(X_0,Y_0)
$X_0=X_1$ $Y_0=Y_1$
$X_0=X_1$ $Y_0=Z_1$
$\overline{\text{female}}(Y_1)$ $\overline{\text{female}}(Z_1)$ $\overline{\text{parent}}(X_1,Y_1)$ $\overline{\text{parent}}(X_1,Z_1)$ sister(Y_1,Z_1)
Y_1=mary
Z_1=julie
Y_1=julie
Z_1=mary
Y_1=mary Z_1=julie
female(mary)
female(julie)
$\overline{\text{sister}}$(mary, julie)

Fig. 1: Basic connection graph with unifications

Resolving on the link between ¬sister(mary, julie) and sister(Y_1, Z_1) produces a new literal, {¬female(mary), ¬female(julie), ¬parent(X_2, mary), ¬parent(X_2, julie)}, which fits into the graph as shown below:



mother(josie, julie)
mother(josie, mary)
$\overline{\text{mother}}(X_0, Y_0)$ parent(X_0,Y_0)
$\overline{\text{female}}(Y_1)$ $\overline{\text{female}}(Z_1)$ $\overline{\text{parent}}(X_1,Y_1)$ $\overline{\text{parent}}(X_1,Z_1)$ sister(Y_1,Z_1)
female(mary)
female(julie)
$\overline{\text{sister}}$(mary, julie)
$\overline{\text{female}}$(mary) $\overline{\text{female}}$(julie) $\overline{\text{parent}}(X_2$, mary) $\overline{\text{parent}}(X_2$, julie)

Fig. 2: Connection graph after one resolution

The major advantage of the CG is that it enables a number of pruning strategies to be used to delete whole sections of the graph, thus cutting the search space down considerably. The reader is referred to (Kowalski 1975) for a discussion of these pruning strategies, and of some of the complexities that arise when the graph contains clauses for recursive axioms; to (Siekmann & Wrightson 1980) for an extension of the method for dealing with equality axioms; and (Eisinger 1986) for results about completeness and consistency of CG proofs. The essential points for current purposes are the basic representation as a graph, and the fact that the only links that resolution introduces into the graph are copies of links that are there already. One final point to remember about the current implementation of the CG is that in addition to the deletion stategies it uses a breadth first search restrained by set of support and unit preference. It is thus complete, in that if a proof exists it will be found, whilst still being reasonably efficient.

3 Decidability

The problem in the example is in fact solvable. It is thus, a fortiori, decidable, with the decision coming down in favour. It is clear, however, that if we had asked various other questions for which a proof was not available the search for a proof would still have definitely terminated. This holds both for questions involving properties and individuals referred to in the axiom set, such as asking whether Josie was Mary's sister, and for ones which mention brand new objects or properties, such as asking whether Alex was Janet's brother. We cannot guarantee termination for all possible questions, as is shown by the following argument. We know that there must be some set of axioms and problem for which the CG algorithm fails to terminate, since it is otherwise a decision procedure for FOPC, which it can't be. We can ensure that this undecidable problem is stated in such a way that there is no overlap between the predicates and individuals mentioned in it and those mentioned in our original problem, so there is no interference. We then construct one big question out of this problem and set of axioms, and ask whether it follows from the axioms in our original problem. By definition, the task of finding out whether the big question follows from the original axioms will lead to a non-terminating search.

We thus see that although we can never guarantee that, for some set of non-degenerate axioms, every possible question will be decided, we can sometimes see right from the start that some particular question will be. We will use the term *decidable theory* for a set of axioms plus a query for which the CG procedure will terminate. The task, then, is to discover properties of connection graphs which indicate that the theory embodied by the graph is decidable, and which are furthermore easy to spot.

We start by making a directed copy of the graph. We do this by a recursive sweep through the graph, starting with literals in clauses in the initial set of support. The direction of links in the copy simply leads away from the set of support. Thus for the example above the ordered graph would look as follows:

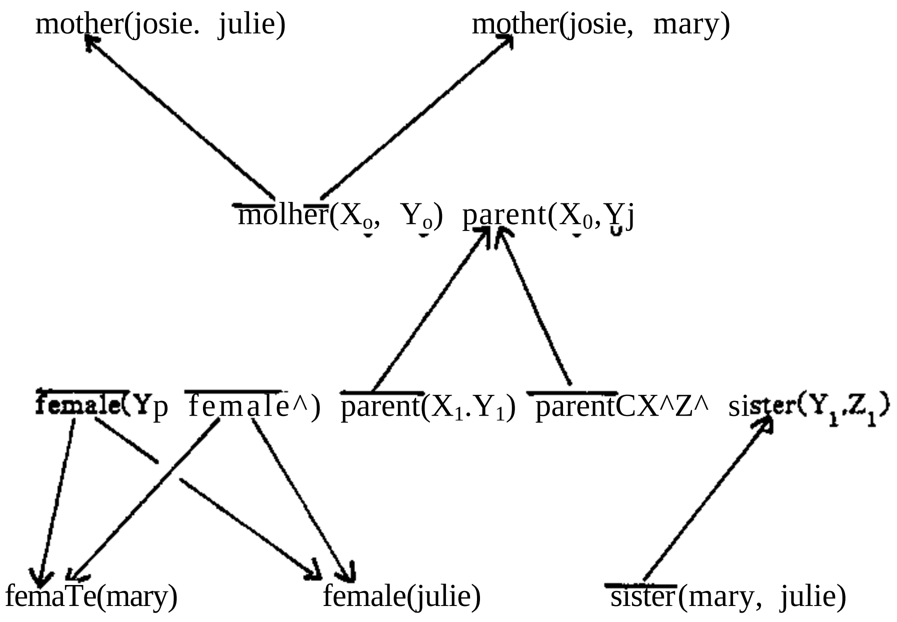

mother(josie. julie)
mother(josie, mary)
molher(X0, Y0) parent(X0,Yj
female(Yp female^) parent(X1.Y1) parentCX^Z^ sister(Y1,Z1)
femaTe(mary)
female(julie)
sister(mary, julie)

Fig. 3: Connection graph with directed links

The striking thing about this graph is that it contains no cycles. This is the first, most obvious, and easiest to calculate indicator of decidability. We will elevate it to the grand status of a theorem:

Theorem 1: if the ordered copy of a connection graph is acyclic then the graph is decidable.

Proof: after a resolution, the resolvant has strictly fewer links than the resolvees pu1 together, since the only links it contains are copies of ones in the resolvees, and one oi these is deleted by the act of resolution. So each of the finite number of links in *tht* original graph can only lead to a finite number of steps.

This is **a** very easy thing to test. The algorithm we use just repeatedly deletes nodes witt no incoming arcs (and hence deletes all their outgoing ones). If it gets to the point where **there are** no nodes left, the original was cycle free, if it gets to the point where there are nodes left but they all have incoming arcs then the original contained cycles. The trouble is **that** very few interesting problems have acyclic graphs. In particular, recursive and mutually recursive axioms introduce cycles (via so-called "pseudo-links"), and so do problems such as the three blocks which require application of the law of the excluded middle for proof.

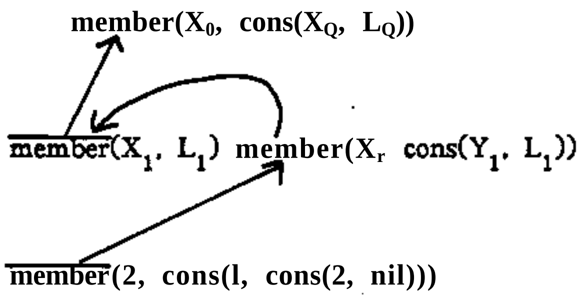
Axioms:

member(XX) -> member(X, cons(YX))

member(X, cons(X.L))

Goal: -'member(2, cons(l, cons(2, nil)))

Example 2: recursive axioms



member(X0, cons(XQ, LQ))
member(X1, L1) member(Xr cons(Y1, L1))
member(2, cons(l, cons(2, nil)))

Fig. 4: Directed graph for recursive problem.

Axioms:

on(a,b) & on(bx) & green(a) & -»green(c)

Goal:

Hx,y(on(x,y) & green(x) & -*green(y))

Example 3: Three blocks problem

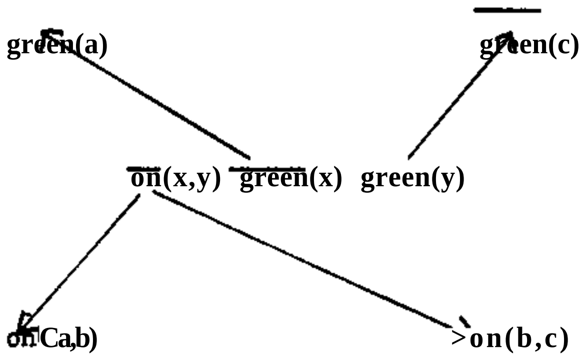


Fig. 5: Directed graph for three blocks problem

We therefore need a sharper tool than simple absence of cycles. If we cannot even show that the system will terminate for these two problems, we are little further on than wher we started. For this we need the following properties of links.

A link is *restrictive* if the associated unifier binds at least one variable at the outward end to a complex term or a constant, but does not bind any variable at the inward end to a complex term. The following are examples of restrictive links:

$\text{member}(x_1, \text{cons}(y_1, l_1)) \longrightarrow \neg\text{member}(x_2, l_2)$ binds l_2 to $\text{cons}(y_1, l_1)$.

$\neg\text{green}(a) \longrightarrow \text{green}(x)$ binds x to a.

Fig. 6: Restrictive links

A link is *destructive* if the associated unifier binds at least one variable to a complex term. Note that links can be destructive even if they also attempt to be restrictive, i.e. if there is a binding of a variable to a term in the other direction. To see an example of a destructive link, just reverse the direction of the arrow in the first of the examples of restrictive ones.

A pair of links is *safe* if (i) the destination of one is the source for the other, and (ii) the unifier associated with the outgoing one mentions no variables not mentioned in the unifie associated with the incoming one. For instance, in Fig. 7

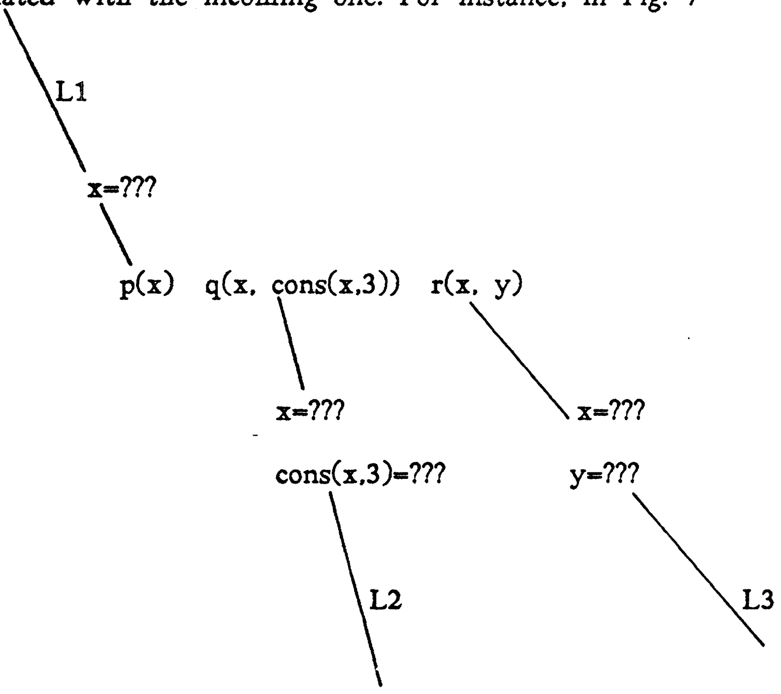


Fig. 7: Safe and unsafe pairs of links

the pair {L1 L2} is safe, but {L1 L3} is not (and trivially nor is {L2 L3}). A chain links is *completely safe* if the successive pairs of links are all individually safe. These a all local, easily computed properties of links. We now have:

Theorem 2: If all cycles are completely safe, contain at least one restrictive link and no destructive ones, the theory is decidable.

Proof: the proof depends on a notion of the *degree* of a cycle. We first define the degree of a term as 0 for a constant, 1 for a variable, and 1 plus the maximum degree of any constituent for a complex term. The degree of a link is then the maximum degree of any term bound by the unifier associated with the link, and the degree of a cycle is the maximum degree of any link in the cycle. Resolving on a restrictive link will always produce a link of lower degree. Resolving on any other non-destructive link will produce a link of the same degree. It is thus clear that if the strategy for selecting links is fair, every cycle in the original graph satisfying the conditions of Theorem 2 must eventually be replaced by one of lower degree, which will in turn be replaced another of lower degree, which will in turn ... Since the initial degree of any term (and therefore of any cycle) must be finite, this process cannot continue indefinitely. Therefore so long as the link selection strategy is fair, any graph satisfying the conditions will be decidable. The strategy we use - breadth first with unit preference and set of support - is indeed fair.

We now have an easily checked property of connection graphs which applies to a large percentage of decidable graphs. It is quite clear that this property cannot be a complete characterisation of decidability, since if it were we would be in a position, by meta-level reasoning, to construct a decision procedure for FOPC. We are left with two open questions. (i) Is the mechanism as it stands worth having? (ii) Should it be made more powerful? Our current feeling is that it does provide extra security with respect to a large enough number of problems to be well worth considering, but that it should not be extended in any way which involves checking non-local properties of chains of links. Proving decidability, though useful, is ancillary to the main task of actually solving the problem, and should not be allowed to use up substantial time and effort that could have been used in attempting to actually derive a proof. A reasonable restriction is that it should certainly not be allowed to become NP-complete itself, as it would if non-local properties of cycles were to become involved.

4 Extending the theorem prover

The arguments above apply to the standard CG algorithm. The theorems provide decidability tests for this algorithm, and the discussion of extensions to these tests concerns the allocation of resources between the tests and the main algorithm. There is, however, another way of thinking about extending the ideas developed above. We could hope that they can be used to make the algorithm reach decisions in more cases. One of the commonest ways for the basic CG algorithm to get into a non-terminating search for a proof is if it is giveii a problem involving symmetric relations, for instance

Axioms:

Vx Vy(married(x, y) -+ married(y, x))

Query: married(janet, John)

Example 4: problem with a symmetric relation

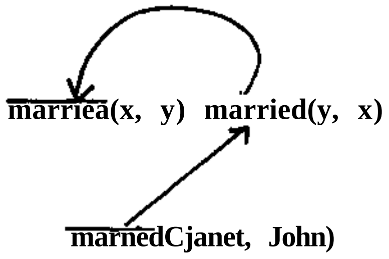


Fig. 8: Undecidable graph

The basic algorithm can be patched to deal with cases like this by including a dynamic check for attempts to resolve on literals which are identical to ones which have been resolved on previously. This works, but it is extremely expensive. It means that the basic act of resolution entails a search through all previous actions. Even worse, since most implementations of the CG method require the variables in the resolvant to be renamed (tc avoid clashes in recursive clauses), the test for whether the current resolution matches a previous one involves a test for unifiability rather than just identity. However, for the common simple case of symmetric links, the test is much simpler. All that you need do for such links is check that when the resolution is performed, the associated unification binds at least one non-variable to a variable. If it does you are not in a loop, if it does not then you are and should hence abandon this line of investigation. We therefore suggest that the static analysis of the graph for satisfaction of the conditions of Theorems 1 and 2 should also include a test for the presence of symmetric links, and that the unification algorithm should inspect such links carefully to ensure that some progress is made when they are resolved upon. This will improve the coverage of the basic CG method, without costing anything at all for cases where there are no symmetric relations, and without costing very much where there are.

5. Conclusions

The main conclusion is that static analysis of the connection graph can provide information which improves the applicability of the CG method. The two particular forms of static analysis that we have described provide a check for termination of the algorithm and a minor extension to deal efficiently with cases for which the basic algorithm would fail to terminate. We do not pretend to have dealt with all possible ways of extracting information about the performance of the CG method by static analysis of the graph. It is sufficient, for now, to show that interesting results can be obtained by comparatively simple analysis. It is important, when performing this sort of pre-processing, not to develop complex and time-consuming algorithms. The overall goal is always to try to prove the given theorem. Static analysis of the graph is only relevant insofar as it gives us either more confidence about our chances of arriving at a a conclusion or, as in the second case, actually makes it more likely that we will do so.

References

Bibel, W. (1982) *Automated theorem proving* Vieweg, Wiesbaden

Boolos, G. & Jeffrey, R.C. (1980) *Computability and logic* Cambridge University Press, Cambridge.

Eisinger. N. (1986) "What you always wanted to know about clause graph resolution" *8th Int. Conf. on Automated Deduction* 316-336, ed. J.Siekman, Springer Verlag

Kowalski, R. (1975) "A proof procedure xzsing connection graphs" *JACM* 22, 572-595

Wallen. L. & Wilson, G. (1987) "A computationally efficient proof system for S5 modal logic" *Advances in artificial intelligence,* eds. Hallam & Mellish, Wiley, Chichester.

---

(1) The three blocks problem: A, B and C are blocks. A is on B, B is on C. A is green, C is not green. Prove that there is a green block on top of a block which is not green. This problem is extremely difficult even to state in a natural way in PROLOG, since any obvious representation requires you to make a negative assertion. It is possible to get a statement by defining a new operator which is equivalent to not but which allows statements of negated facts. With this, the problem can be stated in the natural way, and

a proof will be derived. Unfortunately, the proof will be invalid. The problem is t
linear input resolution does not permit investigation of the excluded middle axiom that B
either green or not green, so most linear input systems would derive a proof on
assumption that B is not green (from negation as failure).