A prolog implementation of the Method of Euler Circles for syllogistic reasoning

Peter Yule
Human Communication Research Centre
University of Edinburgh
2-6 Buccleuch Place
Edinburgh, U.K.
pgy@cogsci.ed.ac.uk

May 27, 1996

Abstract

This paper presents a prolog implementation of syllogistic logic, based on the Individual Identification Algorithm (IIA) (Stenning & Yule 1996), which has been shown to be relevant to human performance in syllogistic reasoning tasks. The IIA is isomorphic to the Method of Euler Circles (Euler 1772), and turns on the identification of necessary individuals prior to drawing quantified conclusions. Three variants of the method are presented: (1) Individual Identification in the case of two premisses, (2) Individual Identification in the case of multiple premisses, and (3) Drawing quantified conclusions in the case of two premisses. The complete prolog code for each variant is included, along with examples.

1 Introduction

Recently, there has been some interest in the Method of Euler Circles as a model of human syllogistic reasoning (Euler 1772; Stenning & Oberlander 1994,1995; Stenning & Yule 1996). Contrary to previous misinterpretations of Euler's method (Erickson 1974, 1978; Guyote & Sternberg 1981), this approach does not lead to a combinatorial explosion of cases, since it distinguishes between necessary and possible individuals by marking regions of the Euler diagrams, in a manner similar to that employed by Mental Models theory (Johnson-Laird 1983, Johnson-Laird & Bara 1984, Byrne & Johnson-Laird 1991).

Moreover, this interpretation of Euler's method has been shown to be functionally indistinguishable from a variety of sentential methods for syllogistic reasoning, which turn on the identification of necessary individuals, so it is one member of a family of implementations of the Abstract Individual Identification Algorithm (Abstract IIA) (Stenning

& Yule 1996, Yule 1996). The `prolog` programs presented here are closely related to the graphical method, insofar as they make the same kinds of information explicit.

This paper presents the code for individual identification in the case of two premisses, which has already appeared briefly in Stenning & Yule (1996), but in this paper it is discussed in greater detail, and extended to cover two different domains: individual identification for multiple syllogistic premisses (a generalisation of the two-premiss case), and the drawing of quantified conclusions in the case of two premisses (the standard categorical syllogism). The first, although perhaps of little practical utility, can be viewed as a didactic device to help illuminate the workings of the two-premiss, 'degenerate' case, and it suggests a possible line of future research in human reasoning. The second constitutes a complete decision procedure for the standard categorical syllogism, illustrating how the Individual Identification Algorithm can underlie competence in this domain, and shows how the term-ordering phenomena associated with the source/conditional distinction can manifest as the Figural Effect (Johnson-Laird & Bara 1984 - see below) in the standard syllogistic reasoning task. It is hoped that the publication of these programs can usefully supplement the material presented by Stenning & Yule (1996), since they are working models of the processes and theories discussed in that paper, but they may even be useful as data-scoring tools for researchers in the field of syllogistic reasoning.

Before presenting the programs, however, it is necessary to summarise the Abstract IIA, and briefly explain its relation to the Method of Euler Circles.

2 The Individual Identification Algorithm

The essence of the Individual Identification Algorithm is that the meanings of premisses are represented in terms of two sets of individuals, the *Minimal model* and the *Maximal model*, where the Minimal model is the set of individuals whose existence is *entailed* by the truth of the premiss, and the Maximal model is the set of individuals whose existence is *consistent* with the truth of the premiss. These are, as their names suggest, respectively the smallest and largest sets of individuals compatible with the truth of the premiss. It should be noted that, in accordance with the standard interpretation of the syllogism dating back to Aristotle (336 BCE/1983), universal statements are taken to have existential consequences, so that a statement such as "No As are Bs" entails the existence of As that are not Bs, and Bs that are not As. Similarly, "All As are Bs" entails the existence of As that are Bs. This is sometimes known as the "no empty sets" axiom.

The IIA then combines the models for each of a pair of premisses to find the Minimal model of the premiss pair, the set of individuals whose existence is entailed by the premisses taken together. These necessary individuals have an intimate relation to the standard quantified conclusions for the premiss pair. If there is a quantified conclusion, then there is a necessary individual; however some premiss pairs entail the existence of necessary individuals without having corresponding quantified conclusions, owing to the expressive limitations of the grammar of syllogistic sentences. For example, "Some Bs are not As, No Bs are Cs" entails the existence of Bs that are neither As nor Cs (as does "No Bs are

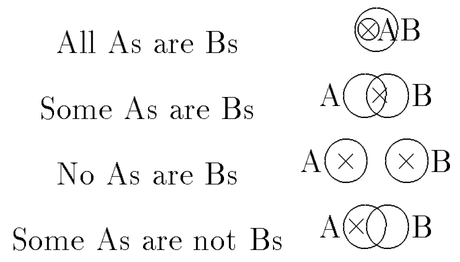

Figure 1: Characteristic diagrams for each premiss type. Regions representing necessary individuals are marked with a $\times$.

As, No Bs are Cs" given the no empty sets axiom), but since the grammar of statements restricts the subjects of conclusions to be positive terms, and neither of the end terms is positive in the specification of the necessary individual, it is strictly ungrammatical to conclude "Some of the not-Cs are not-As", although it is otherwise valid.

The fundamental operation used by the method is unification (Shieber 1986). Necessary individuals can be found by taking the Minimal model (Min) of *one* of the premisses, which we call the *source premiss*, and for each member i of Min, create the set of all unifications of i with each member j of Max, the Maximal model of the other (*conditional*) premiss. If the set of unifications has only one member I, then I is a necessary individual for the premiss pair. To find all members of the Minimal model of the premiss pair, it is necessary to test each member of each premiss' Minimal model in turn.

This method is a straightforward translation of the Method of Euler Circles: each premiss can be represented by a single diagram composed of two circles corresponding to the terms of the premiss (see Figure 1). These represent possible individuals as regions in the diagram, so that the set of regions in a diagram corresponds to the Maximal model of the premiss. The Minimal model of the premiss is represented by marking those regions in the diagram corresponding to necessary individuals.

To find the set of necessary individuals for the premiss pair, take the characteristic diagram for each premiss, and create a new diagram (the *registration diagram*) by identifying the circles corresponding to the middle term. There may be more than one way of doing this consistent with the premisses, but the diagram which is constructed should be the one with the *maximum number of regions* (see Figure 2 for an example). If, in this process, a region which is marked in one of the characteristic diagrams is bisected by the arc of a circle in the registration diagram, then the mark should not be transferred to the registration diagram; otherwise the mark should be transferred. The set of marked regions in the final registration diagram then corresponds to the set of necessary individuals for the premiss pair, i.e. the Minimal model for the premiss pair, and the full set of regions in the registration diagram corresponds to the Maximal model for the premiss pair.

One point of interest is that the distinction between source and conditional premisses can be shown to be closely related to the Figural Effect (Johnson-Laird & Bara 1984) in human syllogistic reasoning; this can be summarised as a tendency for end terms to preserve

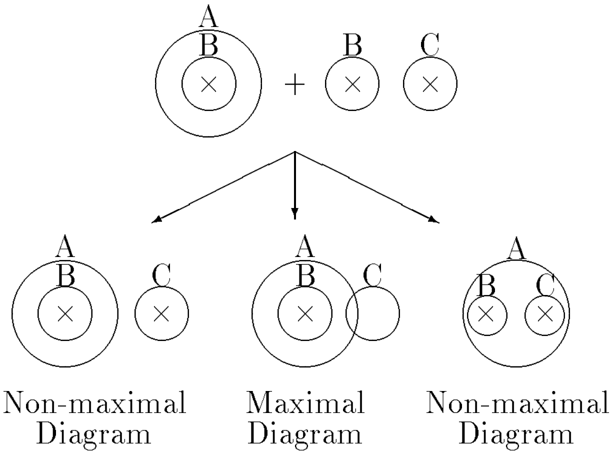

Figure 2: Constructing a registration diagram for the example "All of the Bs are As, None of the Bs are Cs". While any of the three possible registration diagrams is consistent with the premisses, the central one is chosen since it has the largest number of regions.

their grammatical roles from the premisses in which they occur, so that, for example, the overwhelmingly commonest conclusion, given the premisses "All Bs are As, Some Cs are Bs" is "Some Cs are As", despite the fact that "Some As are Cs" is also valid. Stenning & Yule (1996) argue that a model which assumes that terms from source premisses occur first in individual descriptions accounts better for figural variance in term order than does Johnson-Laird & Bara's (1984) working memory model, owing in part to the distribution of source/conditional assignments among the four Figures (see also Yule & Stenning 1992, Yule 1996).

2.1 Individual identification in `prolog`

2.1.1 The case of two premisses

It is quite easy to implement the Individual Identification Algorithm in the logic programming language `prolog` (Clocksin & Mellish 1984). The code assumes standard versions of the predicates `member/2`, `nth/3` and `select/3` (given in Appendix A).

Premiss Representation In prolog, premiss types can be mapped to characteristic representations using the predicate `characteristic/2`, whose first argument is a premiss and whose second is a characteristic representation, implemented as a `Max-Min` pair (see Figure 3). Individuals are represented as lists of features, and models as lists of individuals. Note that each maximal model includes the individual $-A-B$. In graphical terms this corresponds to the background of the corresponding characteristic diagram.

Case Identification The predicate `individual/4` is the top level of the syllogism solution procedure. It retrieves two characteristic representations, one for each premiss, selects

```
characteristic(all(A,B),[[-A,-B],[-A,+B],[+A,+B]]-[[+A,+B]]).
characteristic(some(A,B),[[-A,-B],[-A,+B],[+A,-B],[+A,+B]]-[[+A,+B]]).
characteristic(no(A,B),[[-A,-B],[-A,+B],[+A,-B]]-[[+A,-B],[-A,+B]]).
characteristic(somenot(A,B),[[-A,-B],[-A,+B],[+A,-B],[+A,+B]]-[[+A,-B]]).

individual(Premiss1,Premiss2,Conc,Source) :-
        characteristic(Premiss1,Max1-Min1),
        characteristic(Premiss2,Max2-Min2),
        nth(Source,[Min1,Min2],Min),
        nth(Source,[Max2,Max1],Max),
        member(I,Min),
        unification_test(I,Max,Conc).

unification_test(I,Max,K) :-
        bagof(Unified,J^(member(J,Max),unify(I,J,Unified)),[K]).

unify([],X,X).
unify([H|T],X,[H|NT]) :-
        select(H,X,Y),
        !,
        unify(T,Y,NT).
unify([H|T],X,[H|NT]) :-
        opp(H,Opp),
        not member(Opp,X),
        unify(T,X,NT).

opp(+A,-A).
opp(-A,+A).
```

Figure 3: Prolog code for individual identification

the minimal model (`Min`) from one and the maximal model (`Max`) from the other, selects a critical individual from `Min`, and subjects it to the unification test. If the unification test succeeds, it returns `Conc`, which is instantiated as a necessary individual. Standard prolog backtracking ensures that all critical individuals for both premisses are tested in turn, so the predicate finds all necessary individuals for the premiss pair. Whenever the predicate succeeds, `Source` is incidentally instantiated as the premiss number of the source. If there is no necessary individual the predicate fails immediately.

`unification_test/3` tests the critical individual `I` by attempting to unify it with all members of `Max`. If the `bagof` goal returns a singleton list, only one unification is possible, so that unification denotes a necessary individual.

`unification_test/3` requires a unification predicate `unify/3`, whose first and second arguments are the individuals to be unified, which succeeds with the third argument instantiated as their unification, if this is possible, and fails otherwise. Note that in the event of a successful unification, the features from the first argument appear before any from the second in the result, so that in its use above, terms from the source premiss occur before terms from the conditional premiss in the result.

```
?- individual(some(a,b),all(b,c),Conc,Source).

Conc = [+a,+b,+c]
Source = 1
More (y/n)? y

no
?- individual(all(a,b),no(b,c),Conc,Source).

Conc = [+a,+b,-c]
Source = 1
More (y/n)? y

Conc = [-b,+c,-a]
Source = 2
More (y/n)? y

no
```

Figure 4: Examples of use of the prolog code

Examples and discussion Figure 4 shows two examples of the `prolog` Individual Identification program in action. The first example shows the result of giving it the problem, "Some of the As are Bs, All of the Bs are Cs". The program correctly concludes +A+B+C, and identifies the source as premiss 1. The terms from the source premiss appear first, in order. When backtracking is forced, the program correctly finds no other solutions to the problem.

In the second example, the program is given a problem with two valid individuals. This time, after finding one of the solutions, when backtracking is forced, the program finds the other solution.

To make the operation of the program clearer, it may be useful to supply a Byrd Box Trace of a third example. The problem is the same as the first example in Figure 4, but with premiss order reversed; consequently the program finds the necessary individual only after some backtracking.

```
?- trace,individual(all(b,a),some(c,b),Conc,Source).
   (1) 0 CALL: individual(all(b,a),some(c,b),_426,_460)?
   (2) 1 CALL: characteristic(all(b,a),_698-_702)?
   (2) 1 EXIT: characteristic(all(b,a),[[-b,-a],[-b,+a],[+b,+a]]-[[+b,+a]])
   (3) 1 CALL: characteristic(some(c,b),_906-_910)?
   (3) 1 EXIT: characteristic(some(c,b),[[-c,-b],[+c,-b],[+c,+b],[-c,+b]]-
[[+c,+b]])
```

Since the first two arguments of `individual/4` are instantiated as the two premisses, `all(b,a)` and `some(c,b)`, the first two subgoals of `individual/4` simply retrieve the unique characteristic representations for each of the premisses.

```
   (4) 1 CALL: nth(_460,[[[+b,+a]],[[+c,+b]]],_1162)?skip
 > (4) 1 EXIT: nth(1,[[[+b,+a]],[[+c,+b]]],[[+b,+a]])?
```

Having translated the premisses into characteristic representations, and since the variable `Source` is uninstantiated, one premiss must be selected as the putative source premiss. The standard predicate `nth/3` is used to select one minimal model from a list of two, one for each premiss in order, and to instantiate `Source` as the source premiss number. Since the minimal models are *in premiss order* in the input list, and the minimal model of the first premiss (`[[+b,+a]]`) is selected, `Source` is instantiated as 1.

```
   (5) 1 CALL: nth(1,[[[-c,-b],[+c,-b],[+c,+b],[-c,+b]],[[-b,-a]
,[-b,+a],[+b,+a]]],_1322)?skip
 > (5) 1 EXIT: nth(1,[[[-c,-b],[+c,-b],[+c,+b],[-c,+b]],[[-b,-a]
,[-b,+a],[+b,+a]]],[[-c,-b],[+c,-b],[+c,+b],[-c,+b]])?
```

In order to pair the minimal model of one premiss with the maximal model of the other, the input to the second `nth/3` subgoal is a list of the two premiss' maximal models *in reverse premiss order*. Since `Source` is instantiated, the first maximal model in the list is selected, corresponding to the second premiss. Thus the maximal model is `[[-c,-b],[+c,-b],[+c,+b],[-c,+b]]`.

```
   (6) 1 CALL: member(_1458,[[+b,+a]])?skip
 > (6) 1 EXIT: member([+b,+a],[[+b,+a]])?
```

Since a minimal model may have more than one member (i.e. in the case of 'None'), one of them is chosen as the critical individual. In this example, there is only one, `[+b,+a]`.

```
   (7) 1 CALL: unification_test([+b,+a],[[-c,-b],[+c,-b],[+c,+b],[-c,+b]],_426)?s
 > (7) 1 FAIL: unification_test([+b,+a],[[-c,-b],[+c,-b],[+c,+b],[-c,+b]],_426)?
```

Because there is more than one unification of the critical individual with members of the maximal model, `unification_test/2` fails, forcing backtracking.

```
   (6) 1 REDO: member([+b,+a],[[+b,+a]])?skip
 > (6) 1 FAIL: member(_1458,[[+b,+a]])?
   (5) 1 REDO: nth(1,[[[-c,-b],[+c,-b],[+c,+b],[-c,+b]],[[-b,-a]
,[-b,+a],[+b,+a]]],[[-c,-b],[+c,-b],[+c,+b],[-c,+b]])?skip
 > (5) 1 FAIL: nth(1,[[[-c,-b],[+c,-b],[+c,+b],[-c,+b]],[[-b,-a]
,[-b,+a],[+b,+a]]],_1322)?
   (4) 1 REDO: nth(1,[[[+b,+a]],[[+c,+b]]],[[+b,+a]])?skip
 > (4) 1 EXIT: nth(2,[[[+b,+a]],[[+c,+b]]],[[+c,+b]])?
```

Both the `member/2` and second `nth/3` goals admit no more solutions, so the system backtracks to the first `nth/3` goal, the selection of source premiss and minimal model. This resatisfies, and instantiates `Source` as 2 and `Min` as the minimal model of the second premiss, `[[+c,+b]]`.

```
   (8) 1 CALL: nth(2,[[[-c,-b],[+c,-b],[+c,+b],[-c,+b]],[[-b,-a]
,[-b,+a],[+b,+a]]],_1334)?skip
 > (8) 1 EXIT: nth(2,[[[-c,-b],[+c,-b],[+c,+b],[-c,+b]],[[-b,-a]
,[-b,+a],[+b,+a]]],[[-b,-a],[-b,+a],[+b,+a]])?
```

Since the putative source is now 2, the second `nth/3` goal succeeds and instantiates `Max` as the second member of the list, the maximal model of the first premiss, `[[-b,-a],[-b,+a],[+b,+a]]`. This time, the unification test will succeed, since there is only one member of the maximal model (`[+b,+a]`) with which the only critical individual (`[+c,+b]`) can unify.

```
   (9) 1 CALL: member(_1482,[[+c,+b]])?skip
 > (9) 1 EXIT: member([+c,+b],[[+c,+b]])?
  (10) 1 CALL: unification_test([+c,+b],[[-b,-a],[-b,+a],[+b,+a]],_426)?skip
 >(10) 1 EXIT: unification_test([+c,+b],[[-b,-a],[-b,+a],[+b,+a]],[+c,+b,+a])?
   (1) 0 EXIT: individual(all(b,a),some(c,b),[+c,+b,+a],2)
Conc = [+c,+b,+a]
Source = 2
More (y/n)? y
```

`unification_test/2` is satisfied, and so the top-level goal succeeds; the conclusion is `[+c,+b,+a]` and the source is premiss 2. Backtracking is forced, to search for further valid conclusions, but no subgoals can resatisfy, and so the top-level goal fails (trace omitted).

The program is a complete decision procedure for the Individual Identification task, and it generates individuals with term orders determined by source/conditional attributions (see Stenning & Yule 1996 for discussion of the psychological relevance of this property). When both premisses are source for the same individual, that individual will be generated twice, but with different term orders reflecting the different functional attributions of source and conditional premisses.

2.1.2 The case of multiple premisses

Program code The prolog code can be adapted to the case of multiple syllogistic premisses, as shown in Figure 5. As before, the task is to identify maximally specified necessary individuals, so that each term occurring in the premisses must have a determinate value in the conclusion.

The two predicates `multsyll/3` and `msyll/3` replace the previous top-level predicate `individual/4`. `multsyll/3` is the new top-level predicate, and its first argument is a list of premisses. It selects a premiss from the list to function as source, retrieves a member of the minimal model for this premiss (`Ind`), and passes this, along with the list of the remaining premisses (`Rest`) to `msyll/3`. In the event that `msyll/3` fails, backtracking ensures that each premiss is selected in turn as putative source.

`msyll/3` is a recursive predicate to handle the rest of the premisses. Its first argument is the critical individual entailed by the premisses processed so far, and its second argument is a list of premisses yet to be processed. The first clause defines the base case: if the

```
multsyll(Premisses,Conc,Source) :-
        select(Premiss,Premisses,Rest),
        nth(Source,Premisses,Premiss),
        characteristic(Premiss,_-Min),
        member(Ind,Min),
        msyll(Ind,Rest,Conc).

msyll(Conc,[],Conc).
msyll(Ind,Premisses,MConc) :-
        select(Premiss,Premisses,Rest),
        characteristic(Premiss,Max-_),
        unification_test(Ind,Max,Conc),
        msyll(Conc,Rest,MConc).
```

Figure 5: Prolog code for multiple premisses.

list of premisses is empty, it returns the value of the first argument as the conclusion. In the recursive case, `msyll/3` selects one of the premisses, retrieves its maximal model, and performs the unification test on the input critical individual `Ind`. Provided this is successful, the resulting necessary individual (`Conc`) is passed recursively to `msyll/3` along with the rest of the premisses.

Examples and discussion The operation of `msyll/3` can be summarised as the repeated application of conditional premisses to an input individual derived from the source, which in turn is identified by `multsyll/3`. Thus the multiple-premiss case also crucially depends on the source/conditional distinction. The program can handle premisses in any order: the logic of the problem determines the output term order.

Figure 6 shows two example problems. In the first example, the order of terms in the conclusion is fully determined since the order in which conditional premisses must be applied is determined. As in the two-premiss case, the terms from the source premiss appear first. The second example shows a problem where term order is not fully determined, since there is a choice in the order in which conditionals may be applied. Nevertheless, all conditional applications presuppose the source premiss, so its terms always appear first.

As far as I know, the program is complete for its domain, but since the properties of the domain have not been fully explored, this is at present unproven. It constitutes an extension of the two-premiss program, since it generates exactly the same results as `individual/4` when given a list of two premisses.

3 The standard syllogism - quantified conclusions

Necessary individuals can form the basis for standard quantified syllogistic conclusions. It is easy to draw particular conclusions on the basis of necessary individuals—delete the middle term and choose a positive term as the subject of the conclusion. For example, if

```
?- multsyll([some(a,b),all(b,c),no(c,d)],Conc,Source).

Conc = [+a,+b,+c,-d]
Source = 1
More (y/n)? y

no
?- multsyll([all(a,e),some(a,b),all(b,c),all(c,d)],Conc,Source).

Conc = [+a,+b,+e,+c,+d]
Source = 2
More (y/n)? y

Conc = [+a,+b,+c,+e,+d]
Source = 2
More (y/n)? y

Conc = [+a,+b,+c,+d,+e]
Source = 2
More (y/n)? y

no
```

Figure 6: Multiple premiss examples.

the individual is +A−B−C, delete the middle term feature to give +A−C; the positive term +A forms the subject, and the sign of the remaining term determines the mood, so the conclusion is *Some of the As are not Cs.*

Individual conclusions are not sufficient in themselves to licence universally quantified conclusions, so a further test is necessary. As for particular conclusions, the middle term feature should be deleted and a positive end term feature chosen for the subject, but also, the individual must be the only one in the maximal model of the premiss pair with this value for the subject end term. In graphical terms, the critical region corresponding to the necessary individual must be a whole, unbisected circle. Figure 7 shows an example: in this case the critical region is the 'A' circle, so a universally quantified conclusion with 'As' as the subject is warranted. As with particular conclusions, the sign of the remaining term determines the mood (positive or negative) of the universal conclusion, so in this case, since the remaining feature +C is positive, the conclusion is *All of the As are Cs.*

Although the condition is stated above in terms of the maximal model for the premiss pair, it is not necessary to compute the maximal model to perform the test. This is because the derivation of the necessary individual has already tested for the bisection of the critical region in the registration diagram, so all that is necessary is to determine whether the critical region is circular in the source premiss' characteristic diagram. In `prolog`, this can be achieved with `unification_test/3`: the subject feature (`[+A]` or `[+C]`) is unified with each member of the maximal model of the source premiss, and if only one unification is possible, the universal conclusion is warranted.

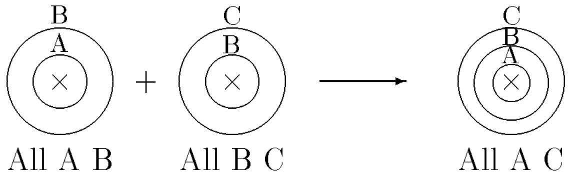

Figure 7: Drawing a universal conclusion—the critical region is the entire 'A' circle (+A+B+C), so a universal conclusion is warranted.

3.1 prolog code for quantified conclusions

Figure 8 shows the additional code required to implement quantified conclusions for pairs of syllogistic premisses. `standard/6` is the top-level predicate, and it can be invoked using a goal like

```
?- standard(all(a,b),all(b,c),Terms,Conc,Syllno,Source).
```

Note that, aside from the subgoal `gensyll/4`, the code for `standard/6` is identical to that for `individual/4` up to the `unification_test/3` subgoal. This is because `standard/6` computes the necessary individual before drawing quantified conclusions. `gensyll/4` is a syntax-checking predicate; it determines the figure of the problem and mood of the premisses, returning these as the structure `Syllno`. This is used for data-scoring in psychological experiments, and has no significance for the conclusion-drawing process. However, `gensyll/4` also instantiates the list `Terms`, which is used by `standard/6` for the identification of the middle term. Moreover, if only `Terms` is instantiated in a call like

```
?- standard(Premiss1,Premiss2,[a,b,c],Conc,Syllno,Source).
```

then `gensyll/4` will generate all possible premiss pairs in turn by backtracking, so that the top-level goal will generate all conclusions for all premiss pairs.

Assuming that `unification_test/3` succeeds in instantiating `NecInd` as a necessary individual, `remove_mid/3` is used to remove the middle term feature, leaving the other two features in the list `EndTerms`. `nth/3` is then used to find the maximal model of the source premiss, `OwnMax`, which is used, along with `EndTerms` as input to `getconc/3`.

`getconc/3` has two clauses, one each for universal and particular conclusions. Each clause's first subgoal selects a positive end-term feature to serve as putative subject for the conclusion, and also instantiates `Fpred`, the predicate feature. In the first, universal case, the positive end-term feature `+Subj` (enclosed in square brackets to denote an individual) is used as input to `unification_test/3`, along with `OwnMax`, to determine whether a universal conclusion is warranted. If so, the predicate `quantifier/5` (also used by `gensyll/4`) is used to instantiate `Conc`, as an 'all' or 'none' conclusion depending on the sign of `Fpred`, and the top-level goal is satisfied. Subsequent backtracking, or failure of the `unification_test/3` goal, results in the invocation of the second, particular

```
standard(Premiss1,Premiss2,[A,B,C],Conc,Syllno,Source) :-
        gensyll(Premiss1,Premiss2,[A,B,C],Syllno),
        characteristic(Premiss1,Max1-Min1),
        characteristic(Premiss2,Max2-Min2),
        nth(Source,[Min1,Min2],Min),
        nth(Source,[Max2,Max1],Max),
        member(I,Min),
        unification_test(I,Max,NecInd),
        remove_mid(B,NecInd,EndTerms),
        nth(Source,[Max1,Max2],OwnMax),
        getconc(EndTerms,OwnMax,Conc).

remove_mid(B,NecInd,EndTerms) :-
        select(+B,NecInd,EndTerms).
remove_mid(B,NecInd,EndTerms) :-
        select(-B,NecInd,EndTerms).

getconc(EndTerms,OwnMax,Conc) :-                /* Universal */
        select(+Subj,EndTerms,[Fpred]),
        unification_test([+Subj],OwnMax,_),
        quantifier(Conc,[Subj,_],u,Fpred,_).
getconc(EndTerms,OwnMax,Conc) :-                /* Particular */
        select(+Subj,EndTerms,[Fpred]),
        quantifier(Conc,[Subj,_],p,Fpred,_).

gensyll(Premiss1,Premiss2,Terms,Syllno-[N1,N2,F]) :-
        figure(F,Terms1,Terms2,Terms),
        quantifier(Premiss2,Terms2,_,_,N2),
        quantifier(Premiss1,Terms1,_,_,N1),
        Syllno is 16*(F-1)+4*N2+N1+1.

quantifier(all(A,B),[A,B],u,+B,0).
quantifier(some(A,B),[A,B],p,+B,1).
quantifier(no(A,B),[A,B],u,-B,2).
quantifier(somenot(A,B),[A,B],p,-B,3).

figure(1,[A,B],[B,C],[A,B,C]).
figure(2,[B,A],[C,B],[A,B,C]).
figure(3,[A,B],[C,B],[A,B,C]).
figure(4,[B,A],[B,C],[A,B,C]).
```

Figure 8: Additional prolog code for quantified conclusions

```
?- standard(all(a,b),all(b,c),_,Conc,_,Source).

Conc = all(a,c)
Source = 1
More (y/n)? y

Conc = some(a,c)
Source = 1
More (y/n)? y

Conc = some(c,a)
Source = 1
More (y/n)? y

no
?- standard(no(a,b),all(b,c),_,Conc,_,Source).

Conc = somenot(c,a)
Source = 1
More (y/n)? y

Conc = somenot(c,a)
Source = 2
More (y/n)? y

no
```

Figure 9: Quantified conclusions examples

clause, which simply selects an end-term feature and generates a particular conclusion using `quantifier/5` in the same manner as the universal case. In the case that both end-term features are positive, backtracking will result in both particular conclusions being generated in turn.

Examples and discussion Figure 9 shows two examples of the quantified conclusions code in action.

The first example shows the problem *All of the As are Bs, All of the Bs are Cs.* The program correctly finds all three valid conclusions and no others, and produces the conclusions whose term order matches the individual's term order (`[+a,+b,+c]`) first.

The second example shows the problem *None of the As are Bs, All of the Bs are Cs.* In this problem, both premisses are source for the same individual, so the conclusion is produced twice, one for each source premiss. Note that although the term order of the underlying individual varies with source premiss (`[-a,+b,+c]` and `[+b,+c,-a]` respectively), the term order of the quantified conclusion does not change. This is the correct behaviour, since the conclusion is not validly convertible.

The program produces all valid conclusions for a given pair of premisses, and if pos-

```
individual(Premiss1,Premiss2,Terms,Conc,Syllno,Source) :-
        gensyll(Premiss1,Premiss2,Terms,Syllno),
        characteristic(Premiss1,Max1-Min1),
        characteristic(Premiss2,Max2-Min2),
        nth(Source,[Min1,Min2],Min),
        nth(Source,[Max2,Max1],Max),
        member(I,Min),
        unification_test(I,Max,Conc).
```

Figure 10: Modifying the `individual` predicate to permit instantiation of premisses using `gensyll/4`.

sible the initial end-term order preserves the term order in the individual due to the source/conditional distinction. However, backtracking will also produce the "counterfigural" conclusions, since the program is a complete decision procedure for the syllogism. When both premisses are source for the same individual, that individual will be generated twice, along with all its associated quantified conclusions.

4 Miscellaneous enhancements

As was mentioned in the previous section, the predicate `gensyll/4` can be used to permit the generation of all possible pairs of premisses in a single run. This feature can easily be extended to the individual identification code for the two-premiss case. To do this, use the definition of `individual/6` given in Figure 10 instead of the previous version of `individual/4`.

All the syllogisms and their conclusions can then be generated in a single run using the goal

```
?- individual(Premiss1,Premiss2,[a,b,c],Conc,Syllno,Source).
```

One possible optimisation of the code is based on the fact that particular premisses never function as conditional premisses, since their maximal models impose no constraint on the maximal model of a premiss pair. Therefore, their maximal models can be dropped entirely from the definitions of the premisses' characteristic representations, as shown below. This allows the programs to run faster without affecting the completeness of the system. Note that since universal premisses can function both as source and conditional premisses, their characteristic representations cannot be simplified in this way.

```
characteristic(all(A,B),[[-A,-B],[-A,+B],[+A,+B]]-[[+A,+B]]).
characteristic(some(A,B),[]-[[+A,+B]]).
characteristic(no(A,B),[[-A,-B],[-A,+B],[+A,-B]]-[[+A,-B],[-A,+B]]).
characteristic(somenot(A,B),[]-[[+A,-B]]).
```

5 General discussion

This paper has shown how to implement the Individual Identification Algorithm (Stenning & Yule 1996) in `prolog`, and how it can be extended to handle both Individual Identification for multiple premisses and the drawing of conventional quantified conclusions from pairs of premisses. It has been shown that term order in the conclusions generated by the programs reflects the source/conditional distinction, by means of the term-ordering properties of the `unify/3` predicate.

The `prolog` implementation differs from the method of Euler Circles insofar as a complete maximal model for the premiss pair is never constructed. Instead, the `prolog` version could be described as focussing on each critical region in turn, and elaborating only those aspects of the maximal model as are relevant to the critical regions. It is this feature which allows it to handle the case of multiple premisses, which cannot be represented in general by single Euler Circle diagrams (Richard Tobin, personal communication).

The association between the source/conditional distinction and term order has been shown to occur in human subjects performing the Individual Identification Task (Yule & Stenning 1992, Stenning & Yule 1996, Yule 1996), and it arguably underlies the well-known Figural Effect in syllogistic reasoning (Johnson-Laird & Bara 1984). While it would be implausible to claim that human reasoners implemented the `prolog` program directly, it has been argued elsewhere that the source/conditional distinction is a higher-order property of syllogistic logic, and can have the same effects in numerous different implementations of the Individual Identification algorithm, in systems as superficially diverse as Euler Circles, natural deduction systems and these `prolog` programs (Stenning & Yule 1996, Yule 1996).

References

Aristotle (336 BCE/1983). Prior Analytics. Tredennik H. (Trans.) Harvard University Press, Cambridge Massachusetts.

Clocksin, W.F. & Mellish, C.S. (1984). *Programming in Prolog.* (2nd edition) Berlin: Springer-Verlag.

Erickson, J.R. (1974). A set analysis theory of behaviour in formal syllogistic reasoning tasks. In R.Solso (Ed.) *Theories in cognitive psychology: the Loyola Symposium.* Potomac: LEA.

Erickson, J.R. (1978). Research on syllogistic reasoning. In R. Revlin & R.E. Mayer (Eds.), *Human Reasoning.* Washington, D.C.: Winston.

Euler, L. (1772). Lettres a une princesse d'Allemagne. *2* sur divers sujets de physique et de philosophie. Letters 102–108.

Guyote, M.J. & Sternberg, R.J. (1981). A transitive-chain theory of syllogistic reasoning. *Cognitive Psychology, 13,* 461-525.

Johnson-Laird, P.N. (1983). *Mental Models.* Cambridge: CUP.

Johnson-Laird, P.N. & Bara, B. (1984). Syllogistic Inference. *Cognition, 16,* 1-61.

Johnson-Laird, P.N. & Byrne, R. (1991). *Deduction.* Hove: LEA.

Shieber, S.M. (1986). *An Introduction to Unification-Based Approaches to Grammar.* Stanford: CSLI.

Stenning, K. & Oberlander, J. (1994). Spatial inclusion as an analogy for set membership: a case study of analogy at work. In K. Holyoak & J. Barnden (Eds.) *Analogical Connections.* Erlbaum.

Stenning, K. & Oberlander, J. (1995). A cognitive theory of graphical and linguistic reasoning: logic and implementation. *Cognitive Science, 19,* 97-140.

Stenning, K. & Yule, P. (1996). Logic and Implementation in Human Reasoning: A Syllogistic Illustration. HCRC Research Paper RP-75, Human Communication Research Centre, University of Edinburgh.

Yule, P. (1996). *Logic and Implementation in Human Reasoning: the Psychology of Syllogisms.* PhD Thesis, University of Edinburgh.

Yule, P. & Stenning, K. (1992). The Figural Effect and a Graphical Algorithm for Syllogistic Reasoning. Proceedings of the 14th Conference of the Cognitive Science Society. Bloomington, Indiana.

A Appendix: standard `prolog` utilities

The following standard `prolog` predicates are assumed by the code in this paper. They may or may not need to be defined, according to the `prolog` implementation used.

```
select(X,[X|Y],Y).
select(Z,[X|Y],[X|W]) :-
        select(Z,Y,W).

member(X,[X|_]).
member(X,[_|Y]) :-
        member(X,Y).

nth(1,[A|B],A).
nth(X,[A|B],C) :-
        nth(D,B,C),
        X is D+1.
```

It may also be necessary to define the predicate `not/1`, or modify the syntax of its occurrence in `unify/3` to suit local conditions.

```
not(X) :-
        call(X),
        !,
        fail.
not(X).
```

Finally, the code assumes that `bagof/3` is available in the local implementation. Unfortunately I don't have a definition for it.