Similarity classes

Master

DEPARTMENT OF ARTIFICIAL INTELLIGENCE

UNIVERSITY OF EDINBURGH

DAI Working Paper: 25

January 1978

Subject: Similarity Classes

Author: Alan Bundy

1. Introduction

The axioms of equality are notorious for causing trouble for theorem provers. The self resolving property of the axioms creates an enormous search space and searches get bogged down in a combinatorial explosion. The standard axiom set is reproduced below (using Kowalski's 1974 notation).

reflexive $X = X \leftarrow$

symmetric $X = Y \leftarrow Y = X$

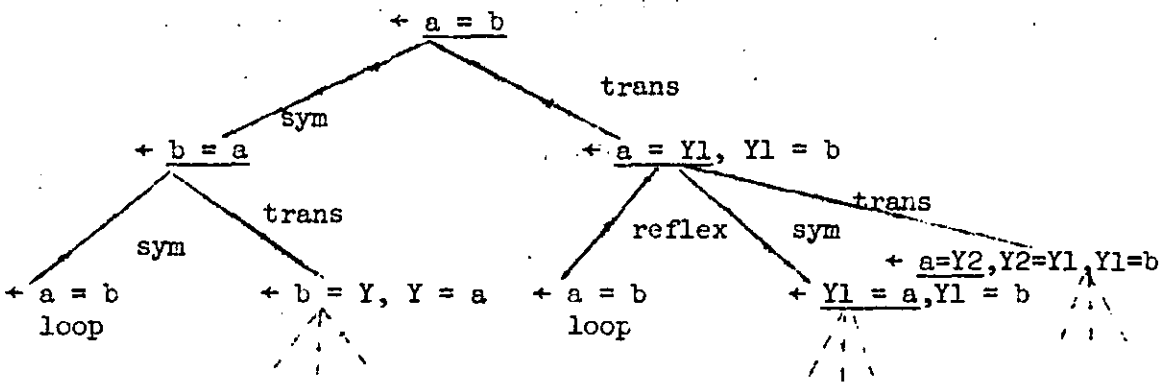
transitive $X = Z \leftarrow X = Y, Y = Z$

substitution(a) $p(X) \leftarrow p(Y), X = Y$

(b) $f(X) = f(Y) \leftarrow X = Y$

where p and f are arbitrary predicates and functions, which may have additional implicit arguments from the ones shown.
(Notation: we use capitals for variables, lower case for constants)

Consider the search space generated by an attempt to prove a = b (using, say, SL resolution, Kowalski & Kuener, 1970):



(the underlined literal is the one selected for further resolving)
Even with the looping branches deleted this is an infinite tree. There is one particularly nasty branch, caused by the repeated application of the transitive law, which produces a sequence of clauses of the form

$\leftarrow a = Yn+1, Yn+1 = Yn, \ldots\ldots, Y1 = b$

There is no way to prune this, because it may indeed be the case that there are n+2 unit assertions of the form

```
a = cn+1 ←
cn+1 = cn ←
.....
.....
c1 = b
```

just waiting for the clause to get the right length. On the other hand, if a and b are unequal, the search will never terminate. If a and b were replaced with proper terms (e.g. f(a,g(b))=f(b,a)) then the substitution(a) axiom would be called into play and the situation would be worse.

2. Equivalence Classes

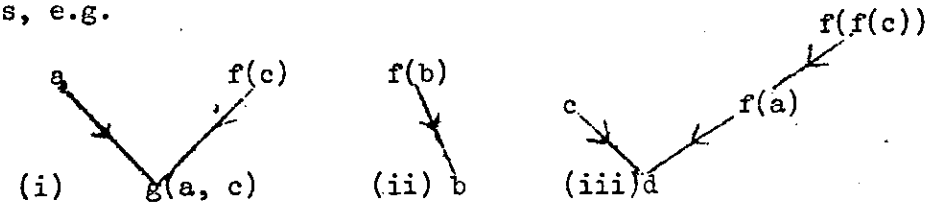
A lot of automatic theorem provers have attempted to avoid this problem by the heuristic use of equivalence classes. We use the word "heuristic", because this technique really only works in the ground case and requires a theoretic investigation to see if its benefit can be made to extend to the general case.

The idea is to keep a set of equivalence classes around for the ground terms (or rather a finite subset of them) e.g.

{a, f(c), g(a, c)}, {b, f(b)}, {c,f(a), f(f(c)), d}

To see whether two ground terms are equal, we just look to see if they are in the same equivalence class. If they are then they are equal. If they are in different classes then they are unequal. Otherwise we do not know.

One convenient way to represent these equivalence classes is as trees, e.g.



To see if two terms are equal we follow the arcs from each term to the (unique) root of the (unique) tree it is in. If these two roots are identical then the terms are equal.

3. Substitution

The above equivalence classes only replace the reflexive, symmetric and transitive laws. They do not replace the substitution laws nor

do they deal with all the ground terms (of which there is generally an infinite number).

One solution to these problems is use the equivalence classes in the unification algorithm, i.e. to unify two ground terms we first look them up in the equivalence classes and if this fails continue the algorithm in the normal way, e.g.

p(f(c)) and p(f(d)) will now unify

because unify(p(f(c)), p(f(d)))

reduces to unify (f(c), f(d)) by unification algorithm.

f(d) is not in an equivalence class so unification is called again producing unify (c, d).

The root of c's tree is d and so is d's tree. Therefore c and d unify.

This does not deal with the general case, since we might hope that

unify(g(X, Y), f(Y))

might succeed binding X to a and Y to c. This will not happen since g(X, Y), f(Y) are not ground, and are not in the equivalence classes.

4. Equality-like Relations

In the MECHO project (Bundy et al, 1977) we have been building a computer program to solve mechanics problems. Part of the program is a problem solver for reasoning about the mechanics world. In the course of designing this we have come across a number of predicates with equality-like properties i.e. they are reflexive, symmetric and transitive. Because of this they each cause combinatorial explosions similar to the equality relation. We have attempted to deal with these explosions by adapting the equivalence class technique.

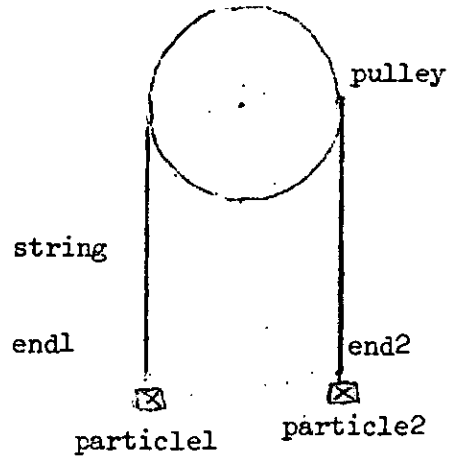
The relations with the equality-like properties are:

concurrent:	a relation between two periods
sameplace:	a relation between two particles or points and a period
factor:	a relation between two units and a number.

We describe each of them in turn.

There are really two versions of "concurrent", distinguished by a third, parity setting, argument. concurrent (X, Y, left) means that the two time periods X and Y have the same first moment. This is true of say 1978 and January 1978, or New Year's Day 1978. concurrent (XY, right) means that X and Y share the same last moment. This is true of the period of a football match and the period of the second half.

sameplace (X,Y,T) means that points X and Y are in contact during the time period or moment T. This is true of each of the ends of the string and each of the particles attached to them in the following diagram.



Or it might be true of an idealization of a train and a platform while the train is in the station, e.g. sameplace (train, platform2, now).

factor (C,X,Y) means that there are C, X units in a Y unit, i.e. factor (60, mins, hrs), factor (36, ins, yds) and factor (1/2240, tons, lbs).

All three of these relations are (after a fashion) reflexive, i.e.

concurrent (X,X, Par) ←

sameplace (X,X,T) ←

factor (1,X,X) ←

and symmetric, i.e.

concurrent (X, Y, Par) ← concurrent (Y,X,Par)

sameplace (X,Y,T) ← sameplace (Y,X,T)

factor (1/C, X,Y) ← factor (C,Y,X)

and transitive, i.e.

concurrent (X,Z,Par) ← concurrent (X,Y,Par),concurrent(Y,Z,Par)

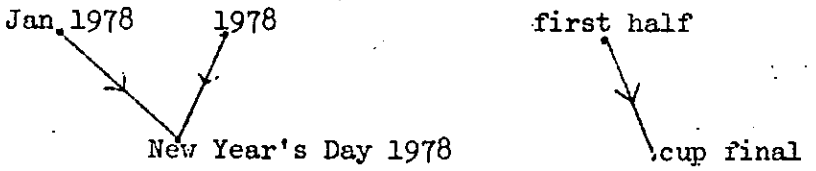
sameplace (X,Z,T) ← sameplace (X,Y,T), sameplace (Y,Z,T)

factor (C1.C2,X,Z) ← factor (C1,X,Y), factor (C2,Y,Z)

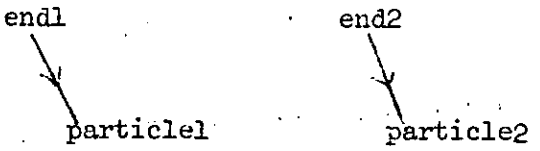
These axioms show that equality is not such a special case after all. Solutions to the so-called "equality problem" should also be applicable to other reflexive, symmetric and transitive relations.

5. Similarity Classes

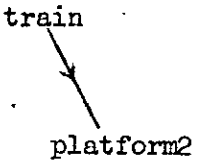
To avoid the combinatorial explosions caused by the relations described above, we have adapted the equivalence class idea. That is, we have built sets of ground terms corresponding to: concurrent left and right periods; objects in fixed contact for each time period and one class for each type of unit, e.g.



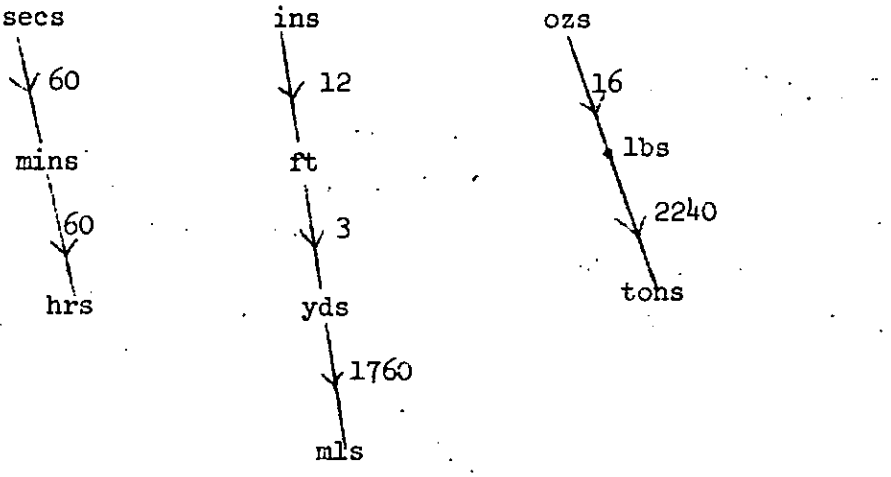
concurrent left classes



sameplace classes for period1



sameplace class for now



factor classes

This representation of relations using similarity classes was implemented by defining the relations with a non-standard set of axioms. Instead of defining the relations using the reflexive, symmetric and transitive laws, their defining axioms specify a search through similarity classes. The axioms were written in PROLOG (Warren, 1977) and contain a certain amount of control information to make sure they are used properly. Because of the similarities between the axioms for each relation it was possible to use a set of general axioms and define concurrent and sameplace as special cases.

Each class tree uses a different kind of arc to link nodes together. This was defined as a relation between nodes. The general similarity class relation "sameclass" takes the arc relation as its third argument - the first two arguments being the two terms whose similarity is to be tested. Thus concurrent and sameplace are defined as:

concurrent (X, Y, left) ← sameclass (X, Y, initseg)

concurrent (X, Y, right) ← sameclass (X, Y, finseg)

sameplace (X, Y, T) ← sameclass (X, Y, touch (T))

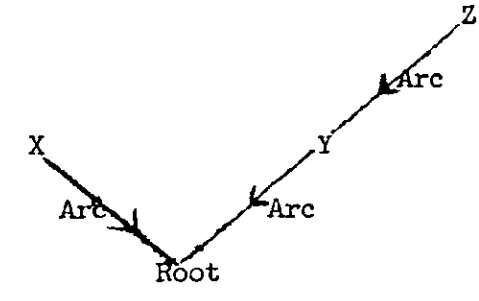
where initseg, finseg and touch(T) are the arc relations for the concurrent trees and the sameplace trees respectively. PROLOG allows us to treat them as terms and also to combine them with their arguments to treat them as goals, e.g. initseg (X, Y), finseg (X, Y) and touch (X, Y, T). The final argument of touch, T, ensures that the sameplace trees for the different times are kept separate.

6. The sameclass algorithm

Sameclass (X, Y, Arc) means that X and Y are in the same Reln-class where Arc is the arc relation of Reln-class.

rep (X, Root, Arc) means that Root is of the Reln-class that X is in. So sameclass is defined in terms of rep, e.e.

sameclass (X, Y, Arc) ← rep (X, Root, Arc), rep (Y, Root, Arc), e.g.



typical Reln-class

The definition of rep (X, Root, Arc) is slightly complicated by efficiency considerations and contains some control information. There are basically two cases. The simple case is when X is already bound to some ground term. Then the Reln-class is uniquely determined and can be traversed from X along the arcs until no more traversal is possible. This version is called toroot and is defined by:

(i) toroot (X, Z, Arc) ← apply (Arc, [X Y]), !, toroot (Y, Z, Arc)

(ii) toroot (X, X, Arc) ←

This definition relies on the PROLOG search strategy which will only use (ii) if it cannot use (i). (i) is applied repeatedly, traversing the tree from X along the Arcs until no further Arcs are found, then (ii) is applied since we must have reached the root. The ! is an evaluable literal which cancels all backtracking on toroot. apply is a second order feature which allows Arc to be applied to its arguments, i.e. apply(Arc, [X, Y]) means Arc(X, Y) but apply (touch (T), [X, Y]) means touch (X, Y, T) (see Bundy & Welham, 1977 for definition).

The other case is when X is not a ground term. In practice, in our program, this situation only occurs when Root is bound to a ground term and X is unbound. The definition in this case is called totips and is designed to work up from the root, returning each node in the tree non-deterministically (i.e. on backtracking):

totips (X, X, Arc) ←

totips (X, Z, Arc) ← apply (Arc, [Y Z]), totips (X, Y, Arc)

totips works in a similar way to toroot.

These two cases are combined using the same PROLOG trick as was used to define toroot.

rep (X, Y, Arc) ← Var (X), !, totips (X, Y, Arc)

rep (X, Y, Arc) ← toroot (X, Y, Arc)

A "pure" definition of rep, stripped of these efficiency considerations, might be:

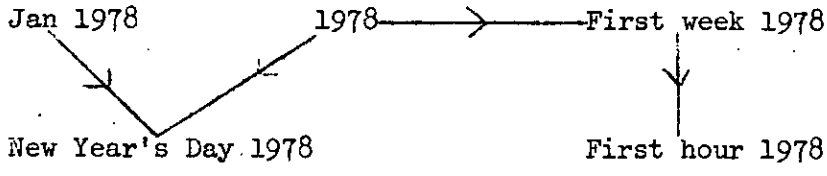
rep (X, X, Arc) ← isroot (X, Arc)

rep (X, Z, Arc) ← ∿ isroot (X, Arc),
Arc (X, Y), rep (Y, Z, Arc)

Note that we have had to mark roots of trees specially, using the predicate isroot and to use 2nd order logic for applying the variable Arc predicate.

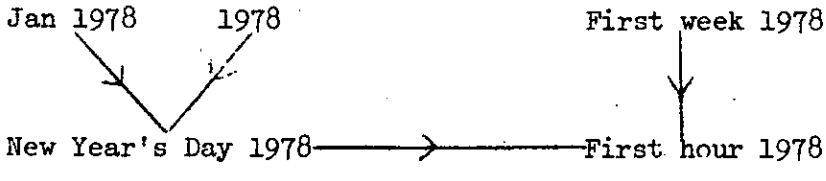
7. Grafting Trees

Discovering whether two terms are in the same class is only half the battle. We also need an algorithm for building the trees. This is done when two terms are asserted to be in the same class. Unfortunately, it is not possible just to draw an arc between them - this could ruin the "treeness" of the similarity classes and disrupt the sameclass algorithm, e.g.



ruined tree - two roots

Suppose two terms are to be put in the same class then the two classes they currently belong to must be grafted. To keep the "treeness" of the grafted classes intact they must be joined at the roots, i.e. one root must be made to point to the other, e.g.



The algorithm for doing this is given below, in PROLOG;

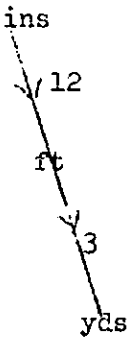
```
graft (X, Y, Arc) ← rep (X, XRoot, Arc) rep (Y, YRoot, Arc)
        cond (diff (XRoot, YRoot), assert_arc (Arc, [XRoot, YRoot]), true).
```

There is no use pretending that this can be read declaritively, as a piece of predicate calculus, since it relies on the side-effect of assert_arc in adding new assertions to the database. It might perhaps best be regarded as a preprocessor, like the procedures for putting formulae in clausal form.

graft works as follows. To assert that X and Y are in the same Reln-class, graft (X, Y, Arc) is called. rep is used twice to find the roots of the classes X and Y belong to, XRoot and YRoot respectively. If XRoot and YRoot are identical nothing happens. Otherwise an arc is drawn between them by asserting Arc (XRoot, YRoot). The PROLOG condition, cond and the non-identity predicate, diff (see Bundy and Welham, 1977) are used to decide between these two cases.

8. Factor Classes

The procedures above were used for concurrent and sameplace. factor was implemented in a very similar way except that the arcs were labelled with the ratio between the units by giving the arc relation, ina, an extra argument, e.g.



represented by ina (12, ins, ft)
ina (3, ft, yds)

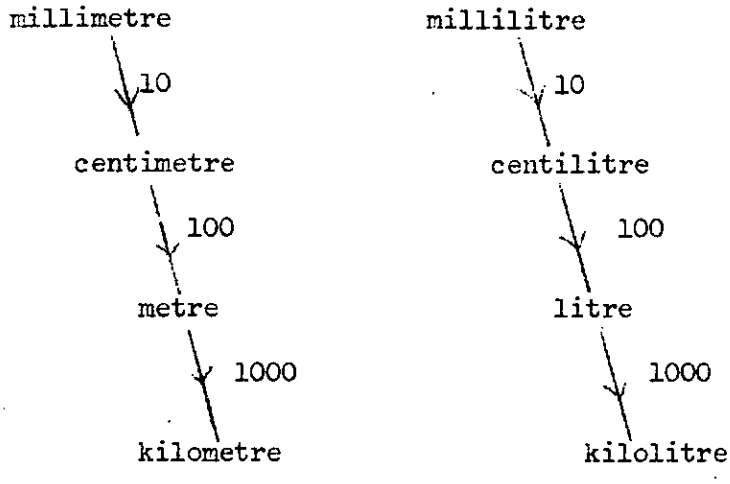
These ratios are multiplied and divided together to keep a running total as the rep and graft algorithms run up and down the trees.

9. Additional Axioms

The sameclass and graft algorithms above replace the reflexive, symmetric and transitive axioms and allow us to convert ground, unit relations into an appropriate form (see appendix for proof). But suppose we have additional axioms for our relation involving several literals or variables. It is no good just asserting these axioms as additional to the sameclass axioms, since the sameclass algorithm expects to find information only in a tree of arc relations. Compare what happens, for instance when arbitrary groundunit sameclass or arc-relation clauses are asserted - in the first case the reflexive, symmetric, transitive axioms are not brought into play - in the second case the sameclass algorithm is disrupted by the "non-treeness" of the similarity classes. Each additional axiom must be incorporated into the similarity class machinery in that same way that graft incorporates ground units. We do not have a general way to do this, nor much faith that a general way can be found. However, we have incorporated a number of axioms on an individual basis and these are described below.

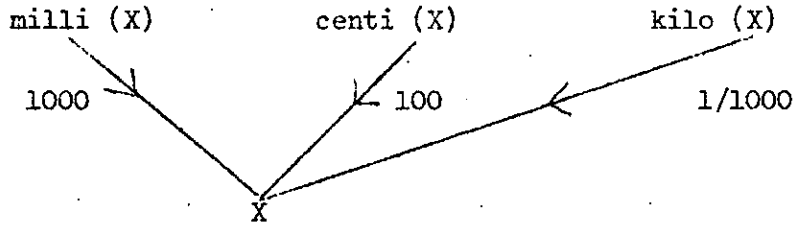
10. Metric Units

The simplest example is an attempt to capture the generalizations behind the naming of the metric units. Using only ground terms we need separate trees for each kind of unit, e.g.



etc.

We can make do with one tree if we make milli, centi, kilo, etc. into functions and allow variables in the tree, e.g.



Our existing sameclass algorithm can handle this tree even though it contains a variable. Essentially we have incorporated into the similarity class machinery non-ground axioms like:

factor (100, centi (X), X)

11. Nested Periods

A more complex example is the building in the sameplace law that "if two objects are in the same place for a period then they are in the same place for a subperiod", i.e.

(1) sameplace (X, Y, SubT) ← subtime (SubT, T), sameplace (X, Y, T)

To see that it is not possible merely to add the axiom to the existing definition of sameplace consider the following situation:

subt trees t trees

where subt is a subperiod of t.
Since sameplace (b, c, t) then sameplace (b, c, subt) and hence sameplace (a, c, subt). However, sameplace (a, c, subt) is unprovable by any combination of axiom (1) and the definition of sameplace using sameclass. For notice that neither sameclass (a, c, touch (subt)) nor sameclass (a, c, touch (t)) are either true or provable.

One way to incorporate axiom (1) into the similarity class machinery is to make sure that the trees for any period are always kept updated with the information from superperiod trees. This can be done by writing a super graft which grafts not only the classes on the present level but all lower levels as well, i.e.

super_graft (X, Y, T) ← graft (X, Y, T),
repeat (subtime(SubT, T) & graft (X, Y, SubT)), !.

(See Bundy and Welham 1977 for definition of repeat).

Using super_graft the trees in our example above become:

subt trees t trees

so that sameplace (a, c, subt) will succeed.
This incorporation of axiom (1) into super_graft is done under the assumption that all information about periods and their subperiods has already been input. Otherwise further updating is required when a new subperiod is created.

12. Substitution

Do the similarity relations also obey the substitution law as they obey the other equality relations, i.e. is it true that:

p (Y) ← concurrent (X, Y, left), p (X)

and sameplace (f(X), f(Y), T) ← sameplace (X, Y, T).

Generally speaking no, but the substitution axioms can hold for a limited set of p's and f's. For instance, the substitution law is true for sameplace and a set of motion predicates, accel, vel, etc. vel(obj, v, dir, T) means that obj has velocity v in direction dir at time T. Provided T is a period the substitution law holds for vel and sameplace, i.e.

vel(Obj2, V, Dir, T) ← sameplace (Obj1, Obj2, T),
period (T), vel(Obj1, V, Dir, T).

This axiom can be incorporated in the manner described in section 3 by modifying the unification algorithm, i.e.

vel(obj1, V, Dir, t) will unify with vel(obj2, V, Dir, t) provided sameplace (obj1, obj2) and period (t). Note that the unification algorithm will have to use this trick only on selected predicates i.e. vel, accel, etc. This is in contrast with the incorporation of the equality substitution in the unification algorithm described in section 3.

This is not such a contrast as it might appear at first sight. Perhaps the equality substitution law should not apply to all predicates. Perhaps it should be possible to designate certain predicates as referentially opaque to which the substitution law does not apply. Consider the substitution law for knows:

knows (X, Z) ← knows (X, Y), Y = Z *

It may be the case that knows (John, 73361) and that telephone_no_of (Bill) = 73361, but we may not want to deduce that knows (John, telephone_no_of (Bill)). Therefore, maybe knows should be designated referentially opaque and the substitution law for knows deleted.

Referentially opaque functions and predicates are rare in mathematics, but they do occur. Perhaps the most well known example is in Gödel numbering. To prove his incompleteness theorems (Gödel, 1931) Gödel defined a function T from terms to numbers (among others). This assigned a unique number to each different term, so it certainly is not the case that:

T (X) = T (Y) ← X = Y *

e.g. x+o = x but T (x+o) ≠ T (x).

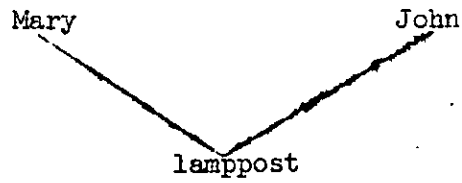
Thus T is referentially opaque.

13. Psychological Plausibility

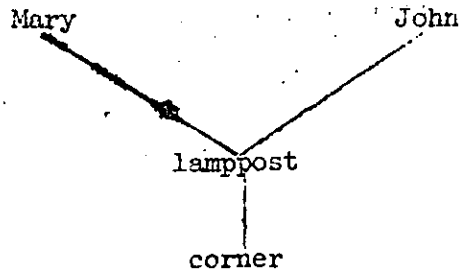
On the basis of limited self observation the similarity class machinery seems to have some plausibility as a model of human reasoning. Certainly using the reflexive, symmetric and transitive axioms of sameplace as described in section 1, would not constitute a model of the human use of the sameplace concept. Humans do not get bogged down in the combinatorial explosions described in section 1, they handle the sameplace concept with ease.

What evidence could we ask for to support the claim of psychological plausibility for similarity classes? We might ask whether classes of similar object appeared to have a natural tree-like order and whether each class tends to have a distinguished member (the root). We might also see whether the various subprocedures of sameclass have a naturally occuring analogue.

All these things are true for sameplace. Of a collection of objects all at the same place the distinguished object is "the place". Two objects are ordered according to which is the more natural candidate for a place - for instance physical objects are preferred to animate ones, parts of the earth to other physical objects etc. For instance, if I tell you that "Mary and John were at the lamppost" the natural representation is:



If I continue "..... on the corner" this changes to

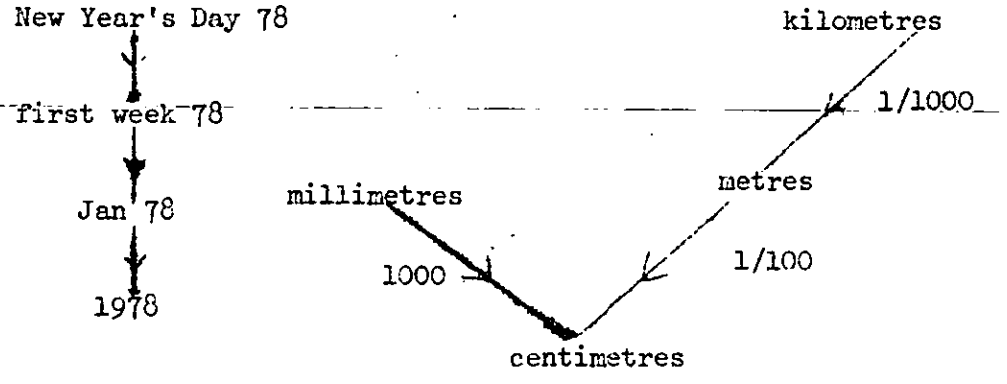


To model this situation, graft would need changing to re-order X and Y before asserting touch (X, Y, T), but this is not a major alteration (see section 7).

Under this scheme isroot is a test to see whether an object is currently being regarded as a "place". rep is a function from

objects to places and corresponds to the normal use of "at" in English. (e.g. Mary is at the corner, but the corner is not at Mary). (N.B. The sameplace concept should not be confused with the "contained-in" concept where an object can be in several places at once, e.g. Mary can be in the street, ... in the city, ...in the country, etc. This should be regarded as an ambiguous use of "place" where a place is not idealized as a point.)

The evidence of psychological plausibility for the concurrent and factor concepts is not so convincing. We have been unable to find natural analogues for the subprocedures rep and isroot. There are natural orderings between the terms and naturally distinguished elements, but this may be coincidental. The most natural ordering for concurrent periods seems to be that of direction with the largest element as the distinguished (root) element. The distinguished element in a class of units is that picked by Physicists as the standard unit, e.g. centimetres, grammes and seconds in the cgs system. The other units are ordered by size relative to this standard.
(N.B. This is at variance with our generalization of the metric names described in section 10. This generalization would force us to use either both metres and grammes or both centimetres and centigrammes as the roots.)



concurrent and factor trees showing "natural" ordering and "natural" roots.

14. Conclusion

The use of similarity classes is an attractive alternative representation for relations definable by reflexive, symmetric and transitive laws. In contrast to an undirected search using these laws the similarity class mechanism is deterministic, terminating and avoids a combinatorial explosion. The drawback of the mechanism is the difficulty of accommodating additional axioms. No general mechanism is known for accommodating them, but it can be done on an individual basis by modifying the tree inputting mechanism (e.g. changing graft) or the unification algorithm.

Much more theoretical work is required to probe the limitations of the similarity class mechanism. Does a general mechanism exist for incorporating new axioms? This seems unlikely - how could the associative law of addition be incorporated in equivalence classes without the termination being lost? If not, under what conditions can axioms be incorporated and how? What is the theoretical relationship between the reflexive, symmetric and transitive laws and the sameclass axioms? Are they equivalent? See the subsequent appendix for the beginnings of an answer to this.

Clearly there remain a lot of unanswered questions!

15. Appendix - Equivalence of Equality Definitions

In this appendix we show that the definition of equality using the similarity class mechanism and definition using the standard axioms of reflexivity, symmetry and transitivity produce equivalent theories. Similar results could easily be derived for sameplace, concurrent and factor using a parallel proof. This result represents the starting point of an attempt to show the adequacy of the similarity mechanism. The difficult part remains of describing and justifying a mechanism for dealing with additional axioms (see section 9).

The proof is divided into two parts:

equivalence class => standard equality

The axioms we will use for equivalence class equality are:

1 equal(X, Y) ← rep(X, R), rep(Y, R).

2 rep(X, X) ← isroot(X).

3 rep(X, R), isroot(X) ← arc(X, Y), rep(Y, R).

Proofs will be by resolution. Theorems to be proved will be preprocessed by (i) negating them, (ii) putting them in clausal form, (iii) applying a "graft" type transformation. The graft transformation consists of marking all ground terms t with isroot(t); carrying out a procedure similar to graft (section 7) except that if XRoot is to be made to point to YRoot then isroot(XRoot) is deleted and ¬isroot(YRoot) is asserted.

reflexive law, Vx(equal(X, Y)): After preprocessing this becomes:

4 isroot(a) ←

5 ← equal(a, a) (where a is new skolem constant)

by resolution in succession we get

6 ← rep(a, R), rep(a, R) (by 5 1)

7 ← isroot(a) (by 6 2 & merging)

8 □ (by 14) Q.E.D.

symmetric law, VX, Y(equal (Y, X) → equal(X, Y)):

after preprocessing

9 arc(b, a) ←

10 isroot(a) ←

11 ← isroot(b)

12 ← equal(a,b)

by resolution

13 ← rep(a,R), rep(b, R) (by 12 & 1)

14 isroot(b) ← rep(a, R), arc(b, Y), rep(Y, R) (by 13, 3)

15 isroot(b) ← arc(b, a), isroot(a) (by 14 2 merging)

16 ← arc(b, a), isroot(a) (by 15 11)

17 ← isroot(a) (by 16 9)

18 □ (by 17 10) Q.E.D.

transitive law VXYZ(equal(X, Y) & equal(Y, Z) → equal(X, Z))

after preprocessing

19 arc(a, b) ←
20 arc(b, c) ←
21 ← isroot(a)
22 ← isroot(b)
23 isroot(c) ←
24 ← equal(a, c)

by resolution

25 ← rep(a, R), rep(c, R) (by 24 1)
26 isroot(a) ← arc(a, Y), rep(Y, R), rep(c, R) (by 25 3)
27 ← arc(a, Y), rep(Y, R), rep(c, R) (by 26 21)
28 ← rep(b, R), rep(e, R) (by 27 19)
29 isroot(b) ← arc(b, Y), rep(Y, R), rep(c, R) (by 28, 3)
30 ← arc(b, Y), rep(Y, R), rep(c, R) (by 29 22)
31 ← rep(c, R) (by 30 20)
32 ← isroot(c) (by 31 2)
33 □ (by 32 23) Q.E.D.

standard equality => equivalence classes

We will use the standard equality axioms

34 equal(X, X) ←
35 equal(X, Y) ← equal(Y, X)
36 equal(X, Z) ← equal(X, Y) equal(Y, Z)

using these we need to prove 1 2 and 3
Since 1 - 3 contain new functions, rep, arc and isroot, we will need to define these. Since rep and arc both imply equality we will define them as equality, i.e.

rep(X, Y) ↔ equal(X, Y)
arc(X, Y) ↔ equal(X, Y)

we will see that isroot can be left undefined.
Using these definitions 1 - 3 become:

1 ' equal(X, Y) ← equal(X, R), equal(Y, R)
2 ' equal(X, X) ← isroot(X)
3 ' equal(X, R), isroot(X) ← equal(X, Y), equal(Y, R)

1 ' yields to a simple application of symmetry 35 and transitivity 36 . We can prove stronger results than 2 ' and 3 ', namely 2 ' and 3 ' without disjuncts containing isroot.

2 " equal(X, Y) ←

3 " equal(X, R) ← equal(X, Y), equal(Y, R)

Clearly 2 " is reflexivity 34 and 3 " is transitivity 36 .

16. References

Bundy, A. and Welham R. 1977a. "Utility Procedures in PROLOG", DAI Occasional Paper No. 9.

Bundy, A., Luger, G., Mellish, C. and Palmer, M., 1977b. "Solving Mechanics Problems: Interim Report to the SRC", DAI Working Paper No. 23.

Gödel, K., 1931. "Über formal unentscheidbare Sätze der Principia Mathematica and verwandter Systeme", Monatsh. Math. Phys. 38, pl73-198.

Kowalski, R. and Kuehner, D., 1970. "Linear Resolution with Selection Function", DCL Memo No. 43.

Kowalski, R., 1974. "Logic for Problem Solving", DCL Memo No. 75.

Warren, D., 1977. "Implementing PROLOG - compiling predicate logic programs", Vol. 1 & 2, DAI Research Reports Nos. 39 and 40.