A Formal Analysis of Case Base Retrieval*

Hugh R. Osborne and Derek G. Bridge
Department of Computer Science
University of York
Heslington
York YO1 5DD

March 21, 1997

Abstract

Case based systems typically retrieve cases from the case base by applying similarity measures. The measures are usually constructed in an ad hoc manner. This report presents a toolbox for the systematic construction of similarity measures. In addition to paving the way to a design methodology for similarity measures, this systematic approach facilitates the identification of opportunities for parallelisation in case base retrieval.

1 Case Memory Systems

In case based reasoning (CBR) a solution to a new problem is proposed by retrieving solutions to similar problems from a set of previously encountered problems and their solutions — a case base. Systems that reason in this way have seen commercial application to a variety of tasks (e.g. help-desk support [19] and autoclave loading [6]).

Generic presentations of CBR are typically "procedural": a number of distinct processing stages are given, and these are connected sequentially, with earlier stages invoking information, and passing it to later stages [17, 1]. The following list of processing stages typify such presentations:

1. The user's statement of the problem is mapped to a representation that allows it to be used for retrieval from, and comparison with cases (problem-solution pairs) in, the case base.
2. This representation is compared with cases in the case base.
3. The best matching cases' descriptions are unlikely to match the problem statement exactly, and therefore the best matching cases' solutions are unlikely to exactly suit the user. So in this stage, the best-matching cases are modified (or *adapted*), adaptations often being suggested by analysis of where the problem statement and the best-matching cases descriptions differ.
4. The revised case is then evaluated. Inadequacies discovered at this stage might result in further adaptation, or a change to the matching criteria.
5. Evaluated solutions can be added to the case base.

This report will concentrate on retrieval mechanisms for case memory systems (stage 2 above).

Retrieval from case based systems is typically achieved by applying a *similarity measure*, σ. The application of the function σ can be realised in a number of ways. Broadly though, approaches can be described as *representational* or *computational* [15]. The representational approach, in its most extreme form, "hard-codes" the similarity function: cases in the case base reside in a data structure, such as a DAG, where

*This research was funded by grant number GR/J99353 in the AIKMS programme of the EPSRC.

proximity in the data structure denotes similarity. The computational approach, on the other hand, will, in its most extreme form, only compute case similarity during retrieval. Representational approaches afford considerable efficiency in retrieval. The data structure is effectively optimised towards retrieval according to the given similarity measure. However, this form of optimisation can lead to a loss of flexibility. It is for this reason that our current work concentrates on the computational approach. In order to address the problem of efficiency, opportunities for parallelisation of the retrieval mechanism must be found. Ironically, parallelism has mostly been investigated for representational approaches (using spreading activation and/or parallel access to the case base through more than one index, e.g. as in the PARADYME system [10]) even though these are already optimised towards their given similarity measure. Indeed, Brown [3] argues that further indexing of the case base may improve efficiency without resorting to parallel implementations. Parallelism for more computational approaches has been much less investigated (but see, e.g., [12]).

Similarity measures have traditionally been constructed on a one off basis by consideration of the problem to be solved. This report considers how similarity measures might be constructed more systematically, their development being driven on two fronts. Firstly, within a given problem area, a similarity measure can be developed incrementally, the latest version of the similarity measure being influenced by insights gained in applying an earlier prototype similarity measure (stage 4 above). Secondly, the construction of the similarity measure can be based on similarity measures constructed for similar problem areas.

Existing CBR systems [2, 16, 7, 11] typically have very simple similarity measures, e.g. a weighted count of the number of exactly matching features, or the reciprocal of a weighted sum of the difference in values of corresponding features. This report considers how similarity measures might be constructed more systematically, generating a partial order over cases. The retrieval mechanism will then return the maxima according to this ordering. Such an approach can be characterised as being "ordinal", rather than "cardinal". This more general approach will be taken in the first half of this report. Later, in section 3, the results from the earlier part of the report will be related to the more traditional numeric (cardinal) approach. A "spin-off" of the work has been the identification of opportunities for parallelisation in case base retrieval.

A case memory system will be considered to consist of a case base and a retrieval mechanism. The case base will be modelled as a finite subset of a set, Θ, of cases, or tuples, equipped with projection functions for accessing the component elements of these cases. A retrieval request is presented to the system as a pair, consisting of an element, ϑ, of Θ and a similarity measure, σ. The case ϑ, known as the *seed*, will, in combination with the similarity measure, represent the "best possible" case. This is in contrast to the traditional approach, in which the seed is the ideal case, and the similarity measure measures only the closeness of retrieved cases to this ideal. In the approach taken here, the similarity measure can, for example, include negation, so that distance from, rather than closeness to, the seed becomes the measure of suitability.

Normally, where cases are problem-solution pairs, one might expect a seed to be a problem, rather than a case. There are, however, situations in which one might like to compare solutions in order to discriminate between cases — for example when consideration of the problem alone has failed to discriminate sufficiently between cases. If only problems are to be compared, σ will be designed to ignore solutions.

The first half of this report — section 2 — will discuss "ordinal" similarity measures, both defining simple "atomic" similarity measures and introducing methods of combining these to form more complex measures. Section 3 will then consider how the results from section 2 can be applied to "cardinal" similarity measures, and how both types of measure may be combined. A model demonstrating some of the results from section 2 has been implemented, and this is presented in section 4. Finally section 5 draws some conclusions.

The report is illustrated throughout by the following example (based on one given by Ryan [18]):

> **Example 1.** Consider the problem of providing a guest with a meal. A meal can be characterised by its name, its ingredients, some indication of its degree of spiciness, the number of calories it contains, etc., etc.
>
> In this example, the ingredients field only indicates if a meal does or does not contain meat. This will be developed further in later examples.

dish::	(**string**,	*the name of the dish*
	bool,	*does it contain meat*
	spiciness,	*how spicy is it*
	N)	*number of calories*

spiciness::= mild| medium mild| medium|
medium hot| hot| extra hot| killer| suicide

The projections $\pi_{\text{ingredients}}$, $\pi_{\text{spiciness}}$ and π_{calories} will select the relevant elements of the list.

Assume that you know how to cook lamb casserole, vegetable biryani, chicken vindaloo and pasta with tomato chilli sauce. These are specified by the following four cases.

name	*ingredients*	*spiciness*	*calories*	
("lamb",	True,	mild,	634	),
("vegetable biryani",	False,	medium hot,	843	),
("chicken vindaloo",	True,	extra hot,	634	),
("pasta",	False,	medium mild,	953	)

Given a guest, called "Mark", who is vegetarian, likes his food hot, and is watching his weight, a seed can be defined expressing these wishes.

$$\mathsf{mark} = (\text{"Mark"}, \text{False}, \mathsf{hot}, 0)$$

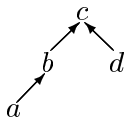
Here the seed on its own represents the "best" possible case, but, as noted earlier, this need not always be so. In general the similarity measure and the seed together define the "best possible" case.

2 Ordinal Similarity Measures

This section presents a repertoire of comparison operations, or similarity measures, and (binary) operations on these similarity measures used to construct new similarity measures. These similarity measures are *ordinal*, i.e. they are symbolic and do not give a numeric value for cases, but order them in terms of their similarity to a seed, i.e. a similarity measure is a function from (features of) cases to partial orders over (features of) cases: $\sigma :: \Theta \to \Theta \to \Theta \to \mathbf{bool}$. Cardinal similarity measures, giving numeric "scores", will be considered in section 3. In this section, sections 2.1 and 2.2 will discuss the construction of similarity measures for individual attributes of cases, while sections 2.3, 2.4 and 2.5 will show how to combine these to form more complex similarity measures for complete cases.

2.1 Atomic Similarity Measures

We define some standard similarity measures below, and illustrate them by an example applied to the set $\{1,2,3,4,5\}$, with the usual total order $\leq$. Note however that the underlying ordering need not be total (see e.g. example 10). The similarity measures are first defined, (e.g. x (flat e) $y = (x = y)4x$ (flat e) $y = (x = y)$, and then an example is given (e.g. flat 3). The illustration is also presented graphically, with arrows representing the ordering relation — e.g. the diagram



represents the relation $\{(a,a),(a,b),(a,c),(b,b),(b,c),(c,c),(d,d),(d,c)\}$.

The similarity measures defined below also generate "increasing" orderings — i.e. for similarity measure σ and seed p the relation x (σ p) y can be interpreted as meaning that x is less similar than y to p.

flat: No elements are related (returns the flat partial order):

$$x \text{ (flat } e) \ y = (x = y)$$

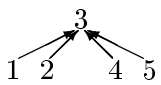
flat 3

1 2 3 4 5

is: Chooses some particular element as being better than all others:

$$x\ (\mathsf{is}\ e)\ y = (y = e) \vee (x = y)$$

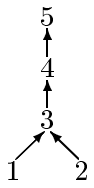
is 3



minimal: The given element is a minimum requirement, and anything greater than that is better:

$$x\ (\mathsf{minimal}\ e)\ y = (x \leq y \wedge e \leq y) \vee (x = y)$$

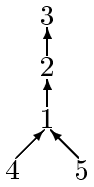
minimal 3



maximal: The given element is a maximum requirement:

$$x\ (\mathsf{maximal}\ e)\ y = (x \leq y \wedge y \leq e) \vee (x = y)$$

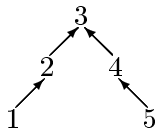
maximal 3



best: The given element is the best ("break the back" of the underlying total order at this element):

$$x\ (\mathsf{best}\ e)\ y = x \leq y \leq e \vee e \leq y \leq x$$

best 3



In addition, the similarity measure id will simply return the underlying ordering, ignoring the the value of the seed.

$$x\ (\mathsf{id}\ e)\ y = (x \leq y)$$

Furthermore the inverse function is defined for similarities, where the inverse of a similarity σ returns the inverse of the order returned by σ.

$$\sigma^{-1}\ e\ x\ y = \sigma\ e\ y\ x$$

Example 2. Recall that the seed for mark was $\mathsf{mark} = (\text{“Mark”}, \text{False}, \mathsf{hot}, 0)$. Assuming that spiciness is ordered from least spicy to most spicy, Mark's preferences are given by: $\pi_{\text{ingredients}}\ \mathsf{is}$, $\pi_{\text{spiciness}}\ \mathsf{best}$ and $\pi_{\text{calories}}\ \mathsf{minimal}^{-1}$, all applied to mark. Note that since 0 is already a minimal element of the type calories the id similarity measure could have been used here rather than $\mathsf{minimal}^{-1}$.

The orders generated on the meals in the case base by each of these atomic measures, applied to the seed mark, are then:

$$\underline{\pi_{\text{ingredients}}\ \mathsf{is}\ \mathsf{mark}}$$
vegetable biryani, pasta
$$\uparrow$$
chicken vindaloo, lamb casserole

The chicken vindaloo and lamb casserole both contain meat ($\pi_{\text{ingredients}}$ chicken vindaloo = $\pi_{\text{ingredients}}$ lamb casserole = True), while the vegetable biryani and pasta do not. Since the corresponding projection of mark is False, the vegetable biryani and pasta are ranked above the chicken vindaloo and lamb casserole.

$$\underline{\pi_{\text{spiciness}}\ \mathsf{best}\ \mathsf{mark}}$$
vegetable biryani chicken vindaloo
$$\uparrow$$
pasta
$$\uparrow$$
lamb casserole

Since there is no case matching the seed exactly for this projection ($\pi_{\text{spiciness}}\ \mathsf{mark} = \mathsf{hot}$), this similarity measure splits the cases into the two disjoint orderings shown here.

$$\underline{\pi_{\text{calories}}\ \mathsf{minimal}^{-1}\ \mathsf{mark}}$$
lamb casserole, chicken vindaloo
$$\uparrow$$
vegetable biryani
$$\uparrow$$
pasta

This is simply the inverse of the ordering imposed by the number of calories in each dish.

Note that no one member of the case base is best in all of these orderings. Sections 2.3, 2.4 and 2.5 will present a method of combining these orderings to arrive at new orderings which may give a (unique) best fit.

2.2 Orders from Other Structures

The orders above (with the exception of flat) were all based on an existing ordering. The general approach was to define a function (e.g. is) which given a seed (e.g. e) would return an ordering (e.g. is e). This section will discuss the derivation of orderings from other structures. A similar approach will be taken — functions will be defined to generate orders from seeds. These functions will make use of auxiliary functions, again applied to seeds, mapping elements of the domain to some ordered set — e.g. $\mathbf{N}$. These auxiliary functions will reflect some notion of distance from the seed, and will usually be written “$\rightsquigarrow$”. These distance functions can also be applied more directly in cardinal similarity measures, as shown in section 3.1.1.

2.2.1 Trees

A tree in which only leaves contain elements is defined by

$$\mathsf{Tree}\ \mathit{elem} ::= \mathsf{Leaf}\ \mathit{elem} \mid \mathsf{Node}\ [\mathsf{Tree}]$$

The distance of an element from a seed can be defined to be the depth of that element in the smallest subtree containing both the element and the seed, though other definitions are possible — for example the sum of the depths of the seed and the element in the smallest subtree containing them both. The range of the distance function is the ordered set $\mathbf{N}_\infty = \mathbf{N} \cup \{\infty\}$. The definition of the distance function $\leftrightsquigarrow$ makes use of three other functions: depth (giving the depth of an element in a tree), $\Subset_{\mathsf{Tree}}$ (the subtrees of a tree), and $\in_{\mathsf{Tree}}$ (which tests if an element appears (as a leaf) in a tree).

The depth of an element in a tree is defined by:

$$
\begin{array}{rcl}
\mathsf{depth}: \text{ gives the depth of an element in a tree} & & \\
\mathsf{depth} :: \mathit{elem} \to \mathsf{Tree}\ \mathit{elem} \to \mathbf{N}_\infty & & \\
\mathsf{depth}\ e\ (\mathsf{Leaf}\ l) & = & 0, \quad \text{if } e = l \\
 & = & \infty, \quad \text{otherwise} \\
\mathsf{depth}\ e\ (\mathsf{Node}\ n) & = & 1 + (\mathsf{min}\ \{\mathsf{depth}\ e\ t \mid t \in n\})
\end{array}
$$

The subtrees of a tree are given by:

$$
\begin{array}{rcl}
\Subset_{\mathsf{Tree}}: \text{ gives the subtrees of a tree} & & \\
\Subset_{\mathsf{Tree}} :: \mathsf{Tree}\ \mathit{elem} \to \{\mathsf{Tree}\ \mathit{elem}\} & & \\
\Subset_{\mathsf{Tree}}\ (\mathsf{Leaf}\ l) & = & \{(\mathsf{Leaf}\ l)\} \\
\Subset_{\mathsf{Tree}}\ (\mathsf{Node}\ n) & = & \{(\mathsf{Node}\ n)\} \cup (\bigcup_{t \in n} \Subset_{\mathsf{Tree}}\ t)
\end{array}
$$

Note that (Node n) above is the complete tree having (Node n) as its root, rather than just the internal node (Node n).

A third function is needed to test if an element appears in a tree:

$$
\begin{array}{rcl}
\in_{\mathsf{Tree}}: \text{ tests if an element is a leaf of a tree} & & \\
\in_{\mathsf{Tree}} :: \mathit{elem} \to \mathsf{Tree}\ \mathit{elem} \to \mathbf{bool} & & \\
e \in_{\mathsf{Tree}} (\mathsf{Leaf}\ l) & = & e = l \\
e \in_{\mathsf{Tree}} (\mathsf{Node}\ n) & = & \exists t \in n : e \in_{\mathsf{Tree}} t
\end{array}
$$

The distance of an element from a seed can now be defined:

$$
\begin{array}{rcl}
\leftrightsquigarrow: \text{ gives the distance of an element from a seed} & & \\
\leftrightsquigarrow :: \mathsf{Tree}\ \mathit{elem} \to \mathit{elem} \to \mathit{elem} \to \mathbf{N}_\infty & & \\
s\ (\leftrightsquigarrow\ t)\ e & = & \mathsf{min}\ \{\mathsf{depth}\ e\ t' \mid t' \in (\Subset_{\mathsf{Tree}}\ t) \wedge s \in_{\mathsf{Tree}} t'\}
\end{array}
$$

Since $s\ (\leftrightsquigarrow\ t)$ clearly defines a function from elements to the ordered set $\mathbf{N}_\infty$, it can be used to define an order generator for trees.

$$
\begin{array}{rcl}
\sqsubset_{\mathsf{Tree}}: \text{ order generator for trees} & & \\
\sqsubset_{\mathsf{Tree}} :: \mathsf{Tree}\ \mathit{elem} \to \mathit{elem} \to (\mathit{elem} \to \mathit{elem} \to \mathbf{bool}) & & \\
e_1\ (\sqsubset_{\mathsf{Tree}}\ t\ s)\ e_2 & = & s\ (\leftrightsquigarrow\ t)\ e_1 \leq s\ (\leftrightsquigarrow\ t)\ e_2
\end{array}
$$

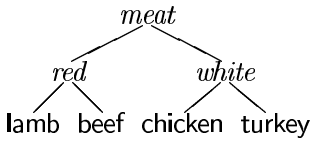
Example 3. Suppose the ingredients field of a case states the type of meat in the dish (ignoring, for the time being, the vegetarian option). A **dish** is then defined to be:

$$
\begin{array}{rll}
\textbf{dish}:: & (\textbf{string}, & \textit{the name of the dish} \\
 & \textbf{meat}, & \textit{the type of meat} \\
 & \textbf{spiciness}, & \textit{how spicy is it} \\
 & \mathbf{N}) & \textit{number of calories} \\
\textbf{spiciness}::= & \mathsf{mild} \mid \mathsf{medium\ mild} \mid \mathsf{medium} & \\
 & \mathsf{medium\ hot} \mid \mathsf{hot} \mid \mathsf{extra\ hot} \mid \mathsf{killer} \mid \mathsf{suicide} & \\
\textbf{meat}::= & \mathsf{lamb} \mid \mathsf{beef} \mid \mathsf{chicken} \mid \mathsf{turkey} &
\end{array}
$$

with the corresponding changes to the non-vegetarian cases:

name	*ingredients*	*spiciness*	*calories*	
("lamb",	lamb,	mild,	634	),
("chicken vindaloo",	chicken,	extra hot,	634	)

Assume now that a tree is defined over **meat** giving a taxonomy of meat types. The tree defined is depicted, with internal nodes labelled for convenience, as:



If Mark's preferred non-vegetarian meal contains turkey, his seed might be:

$$\mathsf{mark} = (\text{"Mark"}, \mathsf{turkey}, \mathsf{hot}, 0).$$

The order generated on the two non-vegetarian cases in the case base by this seed and the given tree (here called mtree) is

$$\frac{(\pi_{\text{ingredients}}\ (\sqsubset_{\mathsf{Tree}}\ \ \mathsf{mtree}))\ \mathsf{mark}}{\begin{array}{c}\text{chicken vindaloo}\\ \uparrow \\ \text{lamb casserole}\end{array}}$$

The distance between turkey and chicken, turkey ($\leftrightsquigarrow$ mtree) chicken, is 1 — this being the minimum depth of tree that contains both turkey and chicken. The distance between turkey and lamb, turkey ($\leftrightsquigarrow$ mtree) lamb, is 2. The chicken vindaloo is therefore preferred over the lamb casserole.

2.2.2 Directed Acyclic Graphs

The functions above have been defined in such a way that they can easily be adapted to apply to DAGs. The auxiliary functions for DAGs will be defined in terms of a general specification of graphs. While the functions could also be applied to more general graphs, extra conditions would probably have to be imposed in any implementation to avoid looping round cycles in such graphs.

A graph is defined to be a quintuple $G = (N, \Sigma_N, \mathcal{L}, \mathcal{E}, r)$, where

- N is the set of nodes
- Σ_N is the node label alphabet
- $\mathcal{L} :: N \to \Sigma_N$ is the node labeling function
- $\mathcal{E} \subseteq 2^{N \times N}$ is the set of edges
- and $r \in N$ is the (unique) root of the graph.

An *element* of a graph is a node label. Graphs will be characterised by their node labels, all the other constituents (N, $\mathcal{L}$, $\mathcal{E}$ and r) will be assumed to be implicitly defined. A graph containing elements of type Σ_N will therefore be an object of type Graph Σ_N.

The depth of an element is given by:

$$\begin{array}{rl} & \mathsf{depth}: \text{ gives the depth of an element in a DAG} \\ \mathsf{depth} :: \Sigma_N \to \mathsf{Graph}\ \Sigma_N \to \mathbf{N}_\infty & \\ \mathsf{depth}\ e\ (N, \Sigma_N, \mathcal{L}, \mathcal{E}, r) & = \ 0, \text{ if } e = \mathcal{L}\ r \\ & = \ 1 + (\mathsf{min}\ \{\mathsf{depth}\ e\ (N, \Sigma_N, \mathcal{L}, \mathcal{E}, m) \mid (r, m) \in \mathcal{E}\}), \text{ otherwise} \end{array}$$

where min $\emptyset$ is defined to be ∞.

The subgraph of G accessible from node n, notation $G[n]$, is defined by:

$$
\begin{array}{rcl}
G[n]: \text{ the subgraph accessible from } n & & \\
G[\cdot] :: N \to \mathsf{Graph}\ \Sigma_N \to \mathsf{Graph}\ \Sigma_N & & \\
G[n]\ (N, \Sigma_N, \mathcal{L}, \mathcal{E}, r) & = & (\{n\} \cup N', \Sigma_N, \mathcal{L}, \{(n,m) \in \mathcal{E}\} \cup \mathcal{E}', n) \\
\text{where} & N' & = \bigcup_{\{m \mid (n,m) \in \mathcal{E}\}} N_m \\
& \mathcal{E}' & = \bigcup_{\{m \mid (n,m) \in \mathcal{E}\}} \mathcal{E}_m \\
(N_m, \Sigma_N, \mathcal{L}, \mathcal{E}_m, m) & = & G[m]\ (N, \Sigma_N, \mathcal{L}, \mathcal{E}, r)
\end{array}
$$

The subgraphs of G can then be defined.

$$
\begin{array}{rcl}
\Subset_{\mathsf{Graph}}: \text{ gives the subgraphs of a graph} & & \\
\Subset_{\mathsf{Graph}} :: \mathsf{Graph}\ \Sigma_N \to \{\mathsf{Graph}\ \Sigma_N\} & & \\
\Subset_{\mathsf{Graph}}\ (N, \Sigma_N, \mathcal{L}, \mathcal{E}, r) & = & \{G[r]\} \cup \bigcup \{\Subset_{\mathsf{Graph}}\ (N, \Sigma_N, \mathcal{L}, \mathcal{E}, m) \mid (r,m) \in \mathcal{E}\}
\end{array}
$$

Again a function is needed to test for membership of a graph.

$$
\begin{array}{rcl}
\in_{\mathsf{Graph}}: \text{ tests if an element is in a graph} & & \\
\in_{\mathsf{Graph}} :: \Sigma_N \to \mathsf{Graph}\ \Sigma_N \to \mathbf{bool} & & \\
e \in_{\mathsf{Graph}}\ (N, \Sigma_N, \mathcal{L}, \mathcal{E}, r) & = & \exists n \in N : \mathcal{L}\ n = e
\end{array}
$$

The distance of an element from a seed, and an order generator for DAGs can now be defined identically to those for trees, with Tree replaced by Graph throughout.

The reader should note that graphs are being used here to define distances between elements and seeds. This is quite distinct from assessing the similarity of two graph structures by some sort of subgraph algorithm, as is found in many case based reasoning systems [15]. There is, however, no reason why such an algorithm could not be used to define an ordering on graphs, and then apply this ordering in the way presented in this report. This properly belongs in the paragraph at the end of this section on user defined types, since the ordering being defined is an ordering on graphs rather than on the elements in the graph.

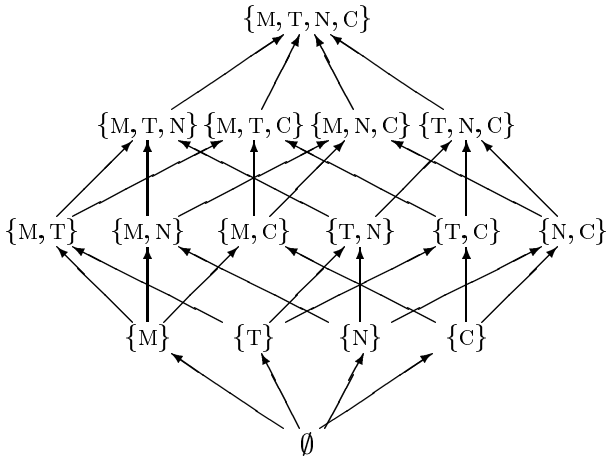
Example 4. The second field in a case may be a set of ingredients.

$$
\begin{array}{rll}
\textbf{dish}:: & (\textbf{string}, & \textit{the name of the dish} \\
& 2^{\textbf{Ingredients}}, & \textit{list of ingredients} \\
& \textbf{spiciness}, & \textit{how spicy is it} \\
& \textbf{N}) & \textit{number of calories} \\
\textbf{spiciness}::= & \mathsf{mild} \mid \mathsf{medium\ mild} \mid \mathsf{medium} \mid & \\
& \mathsf{medium\ hot} \mid \mathsf{hot} \mid \mathsf{extra\ hot} \mid \mathsf{killer} \mid \mathsf{suicide} & \\
\textbf{Ingredients}::= & \text{MEAT} \mid \text{TOMATO} \mid \text{CHILLI} \mid \text{NUTS} \ldots &
\end{array}
$$

with the corresponding changes to the definitions of the cases:

name	*ingredients*	*spiciness*	*calories*	
(“lamb”,	{MEAT,TOMATO},	mild,	634	),
(“vegetable biryani”,	{TOMATO,CHILLI},	medium hot,	843	),
(“chicken vindaloo”,	{MEAT,NUTS,CHILLI},	extra hot,	634	),
(“pasta”,	{TOMATO,CHILLI},	medium mild,	953	)

A directed acyclic graph (in this case a lattice) can be constructed from the subset relation (the names of the ingredients are abbreviated to their initial letters):



Mark can then indicate a preferred set of ingredients:

$$\mathsf{mark} = (\text{“Mark”}, \{\text{TOMATO}, \text{NUTS}, \text{CHILLI}\}, \mathsf{hot}, 0),$$

and an order generated on this DAG (called ingredDAG): $\pi_{\text{ingredients}}\ (\sqsubset_{\mathsf{Graph}}\ \ \mathsf{ingredDAG})$, to define an order over the cases in the case base:

$$\frac{(\pi_{\text{ingredients}}\ (\sqsubset_{\mathsf{Graph}}\ \ \mathsf{ingredDAG}))\ \mathsf{mark}}{\begin{array}{c}\text{vegetable biryani}, \text{chicken vindaloo}, \text{pasta} \\ \uparrow \\ \text{lamb casserole}\end{array}}$$

The ordering generated reflects the fact that the distance function in this graph is a measure of the number of elements in the seed that are not in the set under consideration — i.e.:

$$s\ (\leftrightsquigarrow\ \ d)\ e = |s \setminus e|.$$

2.2.3 Graphs

A similar construction can be used for graphs in general, by defining the distance function to return the length of the shortest path between the seed and an element (or infinity if this path does not exist, possibly because either the seed or the element is not in the graph).

Example 5. A possible option in the running example would be to take Mark out for a meal, rather than entertain him at home. There could be an additional field in the case containing the name of the restaurant at which the meal represented by the case is available. A graph could be given representing the location of the restaurant. Mark could then indicate his preferred location (the restaurant closest to his house, for example) and the ordering would then select those restaurants closest to this choice.

2.2.4 User Defined Types

The same method can be applied to user defined types. A metric should be defined giving the “closeness” of an element to a seed, and this can then be used to define an ordering on that type.

2.3 Boolean Connectives

Sections 2.1 and 2.2 presented a repertoire of similarity measures for individual features of a case. It is now necessary to consider how to combine these similarity measures to form more complex similarity measures for whole cases.

The first obvious candidates for combining orderings are the usual boolean operators. These are covered in this section. The section starts with a presentation of the application of boolean operators to construct

complex similarity measures from simpler ones. This is followed by a discussion of the determination of maxima from these similarity measures. It will then be shown how this can be done in parallel, by transforming similarity measures to a *normal form*.

Sections 2.4 and 2.5 will then introduce other methods of constructing more complex similarity measures, these being *filters*, *priorities* and *preferences* respectively. These new connectives also allow a normal form to be determined, and the computation of the maxima to be executed in parallel.

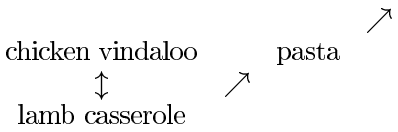
2.3.1 Boolean Operators.

The usual boolean operators ($\wedge$, $\vee$ and $\neg$) may be applied to similarity measures to form new similarity measures. This is done by "lifting" the point-wise boolean operators to operate on similarities. If $\oplus$ is a binary boolean operator then the lifted operator $\widehat{\oplus}$, acting on similarities σ_1 and σ_2, applied to seed p and cases x and y, is defined by

$$(\sigma_1 \mathbin{\widehat{\oplus}} \sigma_2)\ p\ x\ y = (\sigma_1\ p\ x\ y) \oplus (\sigma_2\ p\ x\ y).$$

Example 6. The relation generated by $(\pi_{\text{ingredients}}\ \mathsf{is}\ \widehat{\wedge}\ (\pi_{\text{spiciness}}\ \mathsf{best}\ \widehat{\vee}\ \pi_{\text{calories}}\ \mathsf{minimal}^{-1}))$ when applied to the seed for Mark and the case memory given is:

$$\frac{(\pi_{\text{ingredients}}\ \mathsf{is}\ \widehat{\wedge}\ (\pi_{\text{spiciness}}\ \mathsf{best}\ \widehat{\vee}\ \pi_{\text{calories}}\ \mathsf{minimal}^{-1}))\ \mathsf{mark}}{\text{vegetable biryani}}$$



Note that this is not a partial ordering. The definition of the maxima will be adapted to relations that are not partial orders below.

2.3.2 Determining Maxima.

Note that if $\oplus$ is a boolean operator other than $\wedge$ then, even if $\sigma_1\ p$ and $\sigma_2\ p$ are partial orders, $(\sigma_1 \mathbin{\widehat{\oplus}} \sigma_2)\ p$ is not necessarily a partial order. Since maxima are not defined for arbitrary relations it is necessary to extend the definition of maxima to do this.

The maxima of a partial order $\sqsubset$ are usually defined as

Definition 1

$$\sqcap\ \sqsubset\ S = \{x \in S | \forall y \in S : x \sqsubset y \Rightarrow x = y\}.$$

The usual method for determining maxima of an arbitrary relation $\oplus$ is to take the reflexive transitive closure, $\oplus^*$, of that relation, thus giving a pre-order, and then taking the maxima of the partial order over the equivalence classes generated by the equivalence relation $\otimes = \{(p,q) | p \oplus^* q \wedge q \oplus^* p\}$. I.e. the maxima of a relation are usually defined to be:

Definition 2

$$\begin{array}{rcl} \sqcap\ \oplus\ S & = & \{[x] \in S/\otimes\ |\forall [y] \in S/\otimes : [x] \otimes [y] \Rightarrow [y] \otimes [x]\} \\ & & \text{where } [x] \otimes [y] = x \oplus^* y \end{array}$$

A different approach will be taken here, generalising the concept of maxima to apply to arbitrary relations, avoiding the necessity of generating either the reflexive transitive closure or the equivalence classes.

Since, for a partial order, $(x \sqsubset y \Rightarrow x = y) \equiv (x \sqsubset y \Rightarrow y \sqsubset x)$, definition 1 is equivalent to

Definition 3

$$\sqcap\ \sqsubset\ S = \{x \in S | \forall y \in S : x \sqsubset y \Rightarrow y \sqsubset x\}$$

and this will be taken as the definition of the "maxima" of any relation, i.e. if $\oplus$ is a relation then

Definition 4

$$\sqcap \ \oplus \ S = \{x \in S | \forall y \in S : x \oplus y \Rightarrow y \oplus x\}$$

While the usual definition of the maxima in (2) would give only vegetable biryani as the maximum of the relation in example 6, definition (4) will give both vegetable biryani and chicken vindaloo, which is an equally acceptable interpretation of the relation — there is, after all, no case better than the chicken vindaloo.

Definition (4) has the following properties:

Property 1

$$\sqcap \ (\widehat{\neg} \oplus) = \sqcap \ \oplus^{-1}$$

Proof 1

$$\begin{aligned}
\sqcap \ (\widehat{\neg} \oplus) \ S &= \{x \in S | \forall y \in S : x \ (\widehat{\neg} \oplus) \ y \Rightarrow y \ (\widehat{\neg} \oplus) \ x\} \\
&= \{x \in S | \forall y \in S : \neg(x \oplus y) \Rightarrow \neg(y \oplus x)\} \\
&= \{x \in S | \forall y \in S : y \oplus x \Rightarrow x \oplus y\} \\
&= \{x \in S | \forall y \in S : x \oplus^{-1} y \Rightarrow y \oplus^{-1} x\} \\
&= \sqcap \ \oplus^{-1} \ S.
\end{aligned}$$

Property 2

$$(\sqcap \oplus^1) \ \widehat{\cap} \ (\sqcap \oplus^2) \subseteq \sqcap \ (\oplus^1 \ \widehat{\vee} \ \oplus^2)$$

Proof 2

$$\begin{aligned}
\sqcap \ (\oplus^1 \ \widehat{\vee} \ \oplus^2) \ S &= \{x \in S | \forall y \in S : x \ (\oplus^1 \ \widehat{\vee} \ \oplus^2) \ y \Rightarrow y \ (\oplus^1 \ \widehat{\vee} \ \oplus^2) \ x\} \\
&= \{x \in S | \forall y \in S : x \oplus^1 y \vee x \oplus^2 y \Rightarrow y \oplus^1 x \vee y \oplus^2 x\} \\
&= \{x \in S | \forall y \in S : (x \oplus^1 y \Rightarrow y \oplus^1 x \vee y \oplus^2 x) \wedge (x \oplus^2 y \Rightarrow y \oplus^1 x \vee y \oplus^2 x)\} \\
&= \{x \in S | \forall y \in S : x \oplus^1 y \Rightarrow y \oplus^1 x \vee y \oplus^2 x\} \\
&\quad \cap \qquad (1) \\
&\quad \{x \in S | \forall y \in S : x \oplus^2 y \Rightarrow y \oplus^1 x \vee y \oplus^2 x\} \\
&\supseteq \{x \in S | \forall y \in S : x \oplus^1 y \Rightarrow y \oplus^1 x\} \cap \{x \in S | \forall y \in S : x \oplus^2 y \Rightarrow y \oplus^2 x\} \\
&= (\sqcap \ \oplus^1 \ S) \cap (\sqcap \ \oplus^2 \ S) \\
&= ((\sqcap \oplus^1) \ \widehat{\cap} \ (\sqcap \oplus^2)) \ S.
\end{aligned}$$

Property (1) states that the maxima of the negation of a relation are equivalent to the maxima of the inverse of the relation. This can be useful, since the inverse of a partial order is a partial order, while the negation is not. Property (2) ensures that the intersection of the maxima of two relations will be an acceptable approximation of the maxima of the disjunction of those relations.

A necessary and sufficient condition for the inclusion in property 2 to be an equality is that the two relations involved have a degree of consistency in their inverses. If x is less than y in the first ordering, and greater than y in the second, then it must also be greater than y in the first, and vice versa.

Property 3 *If, and only if, for all x and y in S:*

$$\begin{aligned}
& x \oplus^1 y \wedge y \oplus^2 x \Rightarrow y \oplus^1 x \\
\wedge & \qquad (2) \\
& x \oplus^2 y \wedge y \oplus^1 x \Rightarrow y \oplus^2 x
\end{aligned}$$

then

$$\sqcap \ (\oplus^1 \ \widehat{\vee} \ \oplus^2) \ S = ((\sqcap \oplus^1) \ \widehat{\cap} \ (\sqcap \oplus^2)) \ S$$

Proof 3 (Necessary) *Consider the two relations* $\oplus^1$ *and* $\oplus^2$*, where (only)* $x \oplus^1 y$ *and (only)* $y \oplus^2 x$*. Then* $x \oplus^1 y \wedge y \oplus^2 x \not\Rightarrow y \oplus^1 x$*.*

For these relations $\sqcap \oplus^1 \{x,y\} = \{x\}$ *and* $\sqcap \oplus^2 \{x,y\} = \{y\}$*. The intersection is therefore the empty set. This is not equal to the maxima of* $\oplus^1 \hat{\vee} \oplus^2$*, since* $\sqcap (\oplus^1 \hat{\vee} \oplus^2) \{x,y\} = \{x,y\}$*.*

(Sufficient) *The sufficiency of property (3) follows simply from*

$$(A \wedge B \Rightarrow C) \wedge (A \Rightarrow B \vee C) \equiv A \Rightarrow C,$$

taking $A = x \oplus^i y$*,* $B = y \oplus^j x$ *and* $C = y \oplus^i x$*, for both* $i = 1$*,* $j = 2$ *and* $i = 2$*,* $j = 1$*.*

This allows the inclusion in the proof of property (2) to be replaced by equality.

Taking up the proof of property (2), assuming condition (2), expression (1) can be written as:

$$\begin{aligned}
\sqcap (\oplus^1 \hat{\vee} \oplus^2)\ S \quad &= \quad \{x \in S | \forall y \in S : (x \oplus^1 y \Rightarrow y \oplus^1 x \vee y \oplus^2 x) \wedge (x \oplus^1 y \wedge y \oplus^2 x \Rightarrow y \oplus^1 x)\} \\
&\quad\ \cap \\
&\quad\ \{x \in S | \forall y \in S : (x \oplus^2 y \Rightarrow y \oplus^1 x \vee y \oplus^2 x) \wedge (x \oplus^2 y \wedge y \oplus^1 x \Rightarrow y \oplus^2 x)\} \\
&= \quad \{x \in S | \forall y \in S : x \oplus^1 y \Rightarrow y \oplus^1 x\} \\
&\quad\ \cap \\
&\quad\ \{x \in S | \forall y \in S : x \oplus^2 y \Rightarrow y \oplus^2 x\}
\end{aligned}$$

Since condition (2) holds if $\oplus^1 \hat{\vee} \oplus^2$ is a partial order, then, in this case, the intersection of the maxima will be the maxima of the disjunction.

Property 4 *If* $\oplus^1 \hat{\vee} \oplus^2$ *is a partial order then*

$$\begin{aligned}
& x \oplus^1 y \wedge y \oplus^2 x \Rightarrow y \oplus^1 x \\
\wedge & \\
& x \oplus^2 y \wedge y \oplus^1 x \Rightarrow y \oplus^2 x
\end{aligned}$$

Proof 4 *Condition (2) certainly holds if* $x = y$*, since*

$$\begin{aligned}
& x \oplus^1 x \wedge x \oplus^2 x \Rightarrow x \oplus^1 x \\
& \textit{and} \\
& x \oplus^2 x \wedge x \oplus^1 x \Rightarrow x \oplus^2 x
\end{aligned}$$

If $x \neq y$ *then, since* $(\oplus^1 \hat{\vee} \oplus^2)$ *is a partial order, it cannot be the case that either* $x \oplus^1 y \wedge y \oplus^2 x$ *or* $x \oplus^2 y \wedge y \oplus^1 x$*, since in both cases it would then be the case that* $x\ (\oplus^1 \hat{\vee} \oplus^2)\ y$ *and* $y\ (\oplus^1 \hat{\vee} \oplus^2)\ x$*.*

Since neither of the antecedents in condition (2) can be true, it is trivially true.

These results can be applied to determine the maxima of a relation constructed by application of the boolean operators. This can be done by taking the disjunctive normal form of the boolean expression and determining, in parallel, the maxima of the constituent terms of the normal form, and then taking the intersection.

Example 7. The disjunctive normal form of the similarity measure in example 6 is:

$$(\pi_{\text{ingredients}}\ \mathsf{is}\ \hat{\wedge}\ \pi_{\text{spiciness}}\ \mathsf{best})\ \hat{\vee}\ (\pi_{\text{ingredients}}\ \mathsf{is}\ \hat{\wedge}\ \pi_{\text{calories}}\ \mathsf{minimal}^{-1})$$

The orderings defined by the two terms in this disjunction are:

$$\begin{array}{c}
\underline{(\pi_{\text{ingredients}}\ \mathsf{is}\ \hat{\wedge}\ \pi_{\text{spiciness}}\ \mathsf{best})\ \mathsf{mark}} \\
\text{vegetable biryani} \quad \text{chicken vindaloo} \\
\uparrow \\
\text{pasta} \\
\uparrow \\
\text{lamb casserole}
\end{array}$$

and

$$\frac{(\pi_{\text{ingredients}}\ \mathsf{is}\ \widehat{\wedge}\ \pi_{\text{calories}}\ \mathsf{minimal}^{-1})\ \mathsf{mark}}{\begin{array}{ll}\text{lamb casserole} & \text{vegetable biryani}\\ \updownarrow & \uparrow\\ \text{chicken vindaloo} & \text{pasta}\end{array}}$$

The maxima of these two orderings (which can be computed in parallel) are, respectively, {vegetable biryani, chicken vindaloo} and {lamb casserole, chicken vindaloo, vegetable biryani}, the intersection of which is {vegetable biryani, chicken vindaloo}.

It will be assumed in the following that all similarity measures have been transformed to disjunctive normal form — i.e. that maxima will be calculated by taking the intersections of the maxima of a set of conjunctions of similarity measures:

$$\sqcap\ \sigma\ p\ S = \bigcap_i (\sqcap\ (\bigwedge_j \sigma_{i,j})\ p\ S)$$

Furthermore it will be assumed that each $\sigma_{i,j}$ generates a pre-order. Since the conjunction of pre-orders is a pre-order each term in the normal form will generate a pre-order. In the remainder of this report the symbol σ will be used, unless otherwise stated, to represent a similarity generating a pre-order.

2.4 Filters

Another possibility is to first "filter" the set through some predicate before applying the maximising function, or, alternatively, to first take the maxima, and then apply a filter. Normally a filter will take a predicate over some type, and when applied to a set of that type will give a subset of that type. In keeping with the approach taken in the rest of this report a filter here will take a *relation* over a type, and apply it to a seed to give the predicate that will be applied to a set. The symbol "$\triangleleft$" will be used for filters.

$$\begin{array}{l}\triangleleft :: (\tau \to \tau \to \mathbf{bool}) \to \tau \to \{\tau\} \to \{\tau\}\\ \triangleleft\ \oplus\ p\ S\ \ =\ \ \{x \in S \mid x \oplus p\}\end{array}$$

Example 8. As in example 5 an option could be to take Mark out for a meal, rather than entertain him at home. There could be an additional field in the case containing the price of the meals available. The case base could then be:

name	*ingredients*	*spiciness*	*calories*	*price*
("lamb",	True,	mild,	634	£12),
("vegetable biryani",	False,	medium hot,	843	£10),
("chicken vindaloo",	True,	extra hot,	634	£15),
("pasta",	False,	medium mild,	953	£8)

with the new projection function π_{price} to access the price. You might wish to select the cheapest meal (in combination with requirements relating to other features), while definitely avoiding any meal costing more than £10. The seed for Mark can be defined as:

$$\mathsf{mark} = (\text{"Mark"}, \text{False}, \mathsf{hot}, 0, £10)$$

The cheapest meals will be selected by the similarity measure $\pi_{\text{price}}\ \mathsf{id}^{-1}\ \mathsf{mark}$. Any meals costing more than £10 will be filtered out by the filter $\triangleleft\ (\pi_{\text{price}}\ (\leq))$, so the desired meals would be selected by

$$(\sqcap\ (\pi_{\text{price}}\ \mathsf{id}^{-1}))\ \hat{\cdot}\ (\triangleleft\ (\pi_{\text{price}}\ (\leq)))$$

Filters can be expressed as similarity measures. Given a relation $\oplus$ that is to be applied in a filter, it is possible to define similarity measure $\sigma_\oplus$ such that, except for one special case, $\triangleleft\ \oplus = \sqcap\ \sigma_\oplus$. The exception is when $\triangleleft\ \oplus\ S = \emptyset$, in which case $\sqcap\ \sigma_\oplus\ S = S$.

Property 5 *Let $\oplus$ be any binary relation. Define*

$$\sigma_{\oplus}\ p\ x\ y \quad = \quad y \oplus p.$$

(This similarity measure, $\sigma_{\oplus}$, will be called the "relational similarity measure".)
Then

$$\lhd\ \oplus = \sqcap\ \sigma_{\oplus}.$$

Proof 5

$$\begin{aligned} \sqcap\ \sigma_{\oplus}\ p\ S &= \{x \in S | \forall y \in S : x\ (\sigma_{\oplus}\ p)\ y \Rightarrow y\ (\sigma_{\oplus}\ p)\ x\} \\ &= \{x \in S | \forall y \in S : y \oplus p \Rightarrow x \oplus p\} \end{aligned}$$

Obviously, if there is any y in S such that $y \oplus p$, this is equal to $\{x \in S | x \oplus p\}$, i.e. $\lhd\ \oplus\ p\ S$. If, on the other hand, there is no such y (in which case $\lhd\ \oplus\ p\ S = \emptyset$), it is simply S.

The composition of two filters is equivalent to the conjunction of the two filters.

Property 6

$$(\lhd\ \oplus^1)\ \hat{\cdot}\ (\lhd\ \oplus^2) = \lhd\ (\oplus^1\ \hat{\wedge}\ \oplus^2)$$

Proof 6

$$\begin{aligned} (\lhd\ \oplus^1)\ \hat{\cdot}\ (\lhd\ \oplus^2)\ p\ S &= \lhd\ \oplus^1\ p\ \{x \in S | x \oplus^2 p\} \\ &= \{x \in \{x \in S | x \oplus^2 p\} | x \oplus^1 p\} \\ &= \{x \in S | x \oplus^2 p \wedge x \oplus^1 p\} \\ &= \{x \in S | x\ (\oplus^1\ \hat{\wedge}\ \oplus^2)\ p\} \\ &= \lhd\ (\oplus^1\ \hat{\wedge}\ \oplus^2)\ p\ S \end{aligned}$$

The result of taking the maxima of some similarity measure and then filtering through a relation is the same as the result of taking the intersection of the maxima and the results of the filter.

Property 7

$$(\lhd\ \oplus)\ \hat{\cdot}\ (\widehat{\bigcap_i}\ (\sqcap\ \sigma_i)) = (\lhd\ \oplus)\ \hat{\cap}\ (\widehat{\bigcap_i}\ (\sqcap\ \sigma_i))$$

Proof 7 *The proof is given for a single term. Extending it to an intersection of terms is simple.*

$$\begin{aligned} (\lhd\ \oplus)\ \hat{\cdot}\ (\sqcap\ \sigma)\ p\ S &= \lhd\ \oplus\ p\ \{x \in S | \forall y \in S : x\ (\sigma\ p)\ y \Rightarrow y\ (\sigma\ p)\ x\} \\ &= \{x \in \{x \in S | \forall y \in S : x\ (\sigma\ p)\ y \Rightarrow y\ (\sigma\ p)\ x\} | x \oplus p\} \\ &= \{x \in S | x \oplus p \wedge \forall y \in S : x\ (\sigma\ p)\ y \Rightarrow y\ (\sigma\ p)\ x\} \\ &= \{x \in S | x \oplus p\} \cap \{x \in S | \forall y \in S : x\ (\sigma\ p)\ y \Rightarrow y\ (\sigma\ p)\ x\} \\ &= (\lhd\ \oplus\ p\ S) \cap (\sqcap\ \sigma\ p\ S) \\ &= (\lhd\ \oplus)\ \hat{\cap}\ (\sqcap\ \sigma)\ p\ S \end{aligned}$$

The composition of a maximising function and a filter can be expressed as a composition of a filter and a maximising function.

Property 8

$$(\widehat{\bigcap_i}\ (\sqcap\ \sigma_i))\ \hat{\cdot}\ (\lhd\ \oplus) = (\lhd\ \oplus)\ \hat{\cdot}\ (\widehat{\bigcap_i}\ (\sqcap\ (\sigma_i\ \hat{\wedge}\ \sigma_{\oplus}))).$$

Proof 8 *Again the proof is given for a single term.*

$$\begin{aligned}
(\sqcap\ \sigma)\,\hat{\cdot}\,(\lhd\ \oplus)\ p\ S &= \sqcap\ \sigma\ p\ \{x \in S | x \oplus p\} \\
&= \{x \in \{x \in S | x \oplus p\} | \forall y \in \{x \in S | x \oplus p\} : x\ (\sigma\ p)\ y \Rightarrow y\ (\sigma\ p)\ x\} \\
&= \{x \in S | x \oplus p \wedge \forall y \in S, y \oplus p : x\ (\sigma\ p)\ y \Rightarrow y\ (\sigma\ p)\ x\} \\
&= \{x \in S | x \oplus p \wedge \forall y \in S : y \oplus p \Rightarrow (x\ (\sigma\ p)\ y \Rightarrow y\ (\sigma\ p)\ x)\} \\
&= \{x \in S | x \oplus p \wedge \forall y \in S : (y \oplus p \wedge x\ (\sigma\ p)\ y) \Rightarrow y\ (\sigma\ p)\ x\} \\
&= \{x \in S | x \oplus p \wedge \forall y \in S : (y \oplus p \wedge x\ (\sigma\ p)\ y) \Rightarrow (x \oplus p \wedge y\ (\sigma\ p)\ x)\} \\
&= \{x \in S | x \oplus p\} \cap \\
&\quad \{x \in S | \forall y \in S : (y \oplus p \wedge x\ (\sigma\ p)\ y) \Rightarrow (x \oplus p \wedge y\ (\sigma\ p)\ x)\} \\
&= \{x \in S | x \oplus p\} \cap \\
&\quad \{x \in S | \forall y \in S : (x\ (\sigma_\oplus\ p)\ y \wedge x\ (\sigma\ p)\ y) \Rightarrow (y\ (\sigma_\oplus\ p)\ x \wedge y\ (\sigma\ p)\ x)\} \\
&= \{x \in S | x \oplus p\} \cap \\
&\quad \{x \in S | x\ ((\sigma_\oplus\ p) \mathbin{\hat{\wedge}} (\sigma\ p))\ y \Rightarrow y\ ((\sigma_\oplus\ p) \mathbin{\hat{\wedge}} (\sigma\ p))\ x\} \\
&= \{x \in S | x \oplus p\} \cap \\
&\quad \{x \in S | x\ ((\sigma_\oplus \mathbin{\hat{\wedge}} \sigma)\ p)\ y \Rightarrow y\ ((\sigma_\oplus \mathbin{\hat{\wedge}} \sigma)\ p)\ x\} \\
&= (\lhd\ \oplus)\,\hat{\cdot}\,(\sqcap\ (\sigma \mathbin{\hat{\wedge}} \sigma_\oplus))\ p\ S
\end{aligned}$$

Property 7 can now be applied to remove the composition from this expression.

Property 9

$$(\widehat{\bigcap_i}\ (\sqcap\ \sigma_i))\,\hat{\cdot}\,(\lhd\ \oplus) = (\sqcap\ \sigma_\oplus) \mathbin{\hat{\cap}} (\widehat{\bigcap_i}\ (\sqcap\ (\sigma_i \mathbin{\hat{\wedge}} \sigma_\oplus))).$$

Filters can be used to adjust the results of conjunctions used to express concepts such as "only when" and "except when".

Example 9. Assume Mark likes his food hot, but *only when* it contains meat. Since Mark's seed has False in the ingredients field — i.e. selecting dishes that do *not* contain meat — meals containing meat will be selected by filtering with inequality. Such meals will therefore be selected by

$$(\sqcap\ \pi_{\text{spiciness}}\ \mathsf{best})\,\hat{\cdot}\,(\lhd\ \pi_{\text{ingredients}}\ \neq).$$

Similarly, if he likes his food hot *except when* it contains meat, such meals will be selected by

$$(\sqcap\ \pi_{\text{spiciness}}\ \mathsf{best})\,\hat{\cdot}\,(\lhd\ \pi_{\text{ingredients}}\ =).$$

Filters can also be used to construct more complex preferences. As shown in example 4, the second field in a case might not just indicate, by way of a truth value, if a dish contains meat or not, but be a set of ingredients. Filters can then be used to select dishes containing a minimum (or maximum) set of ingredients, to eliminate dishes containing (or not containing) some specific ingredient, or even, in combination with the operators given in section 2.1, to order cases according to the closeness of their list of ingredients to some ideal.

Example 10. Consider again cases in which the second field is a set of ingredients, as in example 4. If Mark's preferred meal contains nuts, tomato and chilli, but no meat, the seed for Mark will be as defined in example 4:

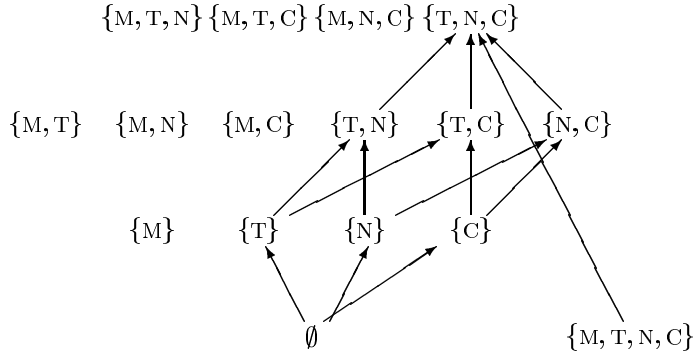
$$\mathsf{mark} = (\text{"Mark"}, \{\text{NUTS}, \text{TOMATO}, \text{CHILLI}\}, \mathsf{hot}, 0)$$

One possible approach would be to take the ordering given by the subset relation, as shown in the lattice in example 4, and to apply the best operator with Mark's set of preferred ingredients

— i.e. to take the maxima $\sqcap \cdot (\pi_{\text{spiciness}}\ \mathsf{best})$, where best now takes the subset ordering $\subseteq$ as the underlying ordering, rather than $\leq$ as in section 2.1.

Note that in example 4 the lattice was used as an example of a directed acyclic graph, with the distance of elements in the DAG from some seed giving an ordering on cases. Here, on the other hand, the lattice is taken to be the underlying ordering on the possible values for the ingredients element of the case (just as $\leq$ was used for **N** in section 2.1).

Applying the best operator with Mark's ideal ingredients will "break the back" of this ordering at {TOMATO,NUTS,CHILLI}, giving:



which, when applied to the four cases in the case base, will return not only the maxima one would expect — the vegetable biryani and the pasta, which both contain tomato and chilli — but will also include the spurious cases chicken vindaloo, which contains meat, nuts and chilli, and lamb casserole, which contains meat and tomato, since the sets of ingredients of these dishes are neither subsets nor supersets of Mark's ideal, and are therefore not related to any other set of ingredients, and are therefore also returned as maxima. A filter can be used to eliminate such cases. The maxima are then given by:

$$(\sqcap \cdot (\pi_{\text{spiciness}}\ \mathsf{best}))\ \hat{\cdot}\ (\triangleleft\ (\pi_{\text{spiciness}}\ (\subseteq \hat{\vee} \supseteq)))$$

The filter (the disjunction of the subset and superset relations) will eliminate any cases for which the list of ingredients is neither a subset nor a superset of Mark's ideal, leaving only those cases in the connected part of the ordering above to be compared by the similarity measure $\pi_{\text{spiciness}}$ best.

An alternative approach is to apply the method in example 4, or to define some metric, as will be discussed in section 3, adding points for each ingredient in Mark's ideal, and deducting points for those not in his ideal. A similar result could be achieved by applying the *preference* connectives, which will be introduced in section 2.5.

2.5 Priorities and Preferences

Another possible type of connective is one which will take one similarity measure as being more significant than another. There are two possible approaches to this. The first applies to the similarity measures themselves, the second to the process of determining maxima. The first of these will be referred to as a *priority* (after [18]), the second as a *preference* (after [5]).

2.5.1 Priorities

The *prioritisation* of relation $\oplus^1$ over relation $\oplus^2$, notation $\oplus^1 \gg \oplus^2$, is a generalisation of lexicographic ordering defined for relations, and is the relation defined by:

Definition 5

$$x\ (\oplus^1 \gg \oplus^2)\ y = (x \oplus^1 y \wedge \neg(y \oplus^1 x)) \vee (x \oplus^1 y \wedge y \oplus^1 x \wedge x \oplus^2 y).$$

The two terms $(x \oplus^1 y \wedge \neg(y \oplus^1 x)$ and $x \oplus^1 y \wedge y \oplus^1 x \wedge x \oplus^2 y)$ in this disjunction satisfy condition 2 on page 11, since both antecedents in this condition will be false. As a consequence, the intersections of the maxima of the two terms will be equal to the maxima of the the prioritisation.

A prioritisation of similarity measures is a prioritisation of relations "lifted" to similarity measures:

$$(\sigma_1 \gg \sigma_2)\ p = (\sigma_1\ p) \gg (\sigma_2\ p).$$

Property 10 *When taking maxima of priorities the first term* $(x \oplus^1 y \wedge \neg(y \oplus^1 x))$ *may be replaced by* $x \oplus^1 y$, *since* $x \oplus^1 y \wedge \neg(y \oplus^1 x) \Rightarrow y \oplus^1 x \wedge \neg(x \oplus^1 y)$ *is equivalent to* $x \oplus^1 y \Rightarrow y \oplus^1 x$.

The prioritisation of similarity σ_1 over similarity σ_2 will therefore be defined as:

Definition 6

$$\sigma_1 \gg \sigma_2 = \sigma_1 \mathbin{\widehat{\vee}} (\sigma_1 \mathbin{\widehat{\wedge}} \sigma_1^{-1} \mathbin{\widehat{\wedge}} \sigma_2).$$

thus avoiding the need for negation.

Property 11 *Prioritisation distributes to the right over disjunction and, when taking maxima, if the two similarity measures being disjoined satisfy condition 2, also to the left.*

$$\sigma_1 \gg (\sigma_2 \mathbin{\widehat{\vee}} \sigma_3) = (\sigma_1 \gg \sigma_2) \mathbin{\widehat{\vee}} (\sigma_1 \gg \sigma_3) \tag{3}$$

and

$$\begin{array}{ll} & (x\ (\sigma_1\ p)\ y \wedge y\ (\sigma_2\ p)\ x \Rightarrow y\ (\sigma_1\ p)\ x) \wedge (x\ (\sigma_2\ p)\ y \wedge y\ (\sigma_1\ p)\ x \Rightarrow y\ (\sigma_2\ p)\ x) \\ \Rightarrow & \\ & \sqcap\ ((\sigma_1 \mathbin{\widehat{\vee}} \sigma_2) \gg \sigma_3)\ p\ S = \sqcap\ ((\sigma_1 \gg \sigma_3) \mathbin{\widehat{\vee}} (\sigma_2 \gg \sigma_3))\ p\ S. \end{array} \tag{4}$$

Proof 11 *The proof of (3) is simple.*

$$\begin{array}{rcl} \sigma_1 \gg (\sigma_2 \mathbin{\widehat{\vee}} \sigma_3) & = & \sigma_1 \mathbin{\widehat{\vee}} (\sigma_1 \mathbin{\widehat{\wedge}} \sigma_1^{-1} \mathbin{\widehat{\wedge}} (\sigma_2 \mathbin{\widehat{\vee}} \sigma_3)) \\ & = & \sigma_1 \mathbin{\widehat{\vee}} (\sigma_1 \mathbin{\widehat{\wedge}} \sigma_1^{1} \mathbin{\widehat{\wedge}} \sigma_2) \mathbin{\widehat{\vee}} \sigma_1 \mathbin{\widehat{\vee}} (\sigma_1 \mathbin{\widehat{\wedge}} \sigma_1^{-1} \mathbin{\widehat{\wedge}} \sigma_3) \\ & = & (\sigma_1 \gg \sigma_2) \mathbin{\widehat{\vee}} (\sigma_1 \gg \sigma_3) \end{array}$$

The proof of (4) is as follows:

$$\begin{array}{rcl} (\sigma_1 \gg \sigma_2) \mathbin{\widehat{\vee}} \sigma_3 & = & (\sigma_1 \mathbin{\widehat{\vee}} \sigma_2) \mathbin{\widehat{\vee}} ((\sigma_1 \mathbin{\widehat{\vee}} \sigma_2) \mathbin{\widehat{\wedge}} (\sigma_1 \mathbin{\widehat{\vee}} \sigma_2)^{-1} \mathbin{\widehat{\wedge}} \sigma_3) \\ & = & \sigma_1 \mathbin{\widehat{\vee}} \sigma_2 \mathbin{\widehat{\vee}} ((\sigma_1 \mathbin{\widehat{\vee}} \sigma_2) \mathbin{\widehat{\wedge}} (\sigma_1^{-1} \mathbin{\widehat{\vee}} \sigma_2^{-1}) \mathbin{\widehat{\wedge}} \sigma_3) \\ & = & \sigma_1 \mathbin{\widehat{\vee}} \sigma_2 \mathbin{\widehat{\vee}} (((\sigma_1 \mathbin{\widehat{\wedge}} \sigma_1^{-1}) \mathbin{\widehat{\vee}} (\sigma_1 \mathbin{\widehat{\wedge}} \sigma_2^{-1}) \mathbin{\widehat{\vee}} (\sigma_2 \mathbin{\widehat{\wedge}} \sigma_1^{-1}) \mathbin{\widehat{\vee}} (\sigma_2 \mathbin{\widehat{\wedge}} \sigma_2^{-1})) \mathbin{\widehat{\wedge}} \sigma_3) \end{array}$$

Since condition 2 holds — i.e. $\sigma_1 \mathbin{\widehat{\wedge}} \sigma_2^{-1} \Rightarrow \sigma_1^{-1}$ *and* $\sigma_2 \mathbin{\widehat{\wedge}} \sigma_1^{-1} \Rightarrow \sigma_2^{-1}$, *the terms* $\sigma_1 \mathbin{\widehat{\wedge}} \sigma_2^{-1}$ *and* $\sigma_2 \mathbin{\widehat{\wedge}} \sigma_1^{-1}$ *are equivalent to* $\sigma_1 \mathbin{\widehat{\wedge}} \sigma_2^{-1} \mathbin{\widehat{\wedge}} \sigma_1{}^{-}1$ *and* $\sigma_2 \mathbin{\widehat{\wedge}} \sigma_1^{-1} \mathbin{\widehat{\wedge}} \sigma_2^{-1}$ *respectively. Therefore:*

$$(\sigma_1 \gg \sigma_2) \mathbin{\widehat{\vee}} \sigma_3 \quad = \quad \sigma_1 \mathbin{\widehat{\vee}} \sigma_2 \mathbin{\widehat{\vee}} (((\sigma_1 \mathbin{\widehat{\wedge}} \sigma_1^{-1}) \mathbin{\widehat{\vee}} (\sigma_1 \mathbin{\widehat{\wedge}} \sigma_2^{-1} \mathbin{\widehat{\wedge}} \sigma_1^{-1}) \mathbin{\widehat{\vee}} (\sigma_2 \mathbin{\widehat{\wedge}} \sigma_1{}^{-}1 \mathbin{\widehat{\wedge}} \sigma_2^{-1}) \mathbin{\widehat{\vee}} (\sigma_2 \mathbin{\widehat{\wedge}} \sigma_2^{-1})) \mathbin{\widehat{\wedge}} \sigma_3)$$

Trivially, $(\sigma_1 \mathbin{\widehat{\wedge}} \sigma_1^{-1}) \mathbin{\widehat{\vee}} (\sigma_1 \mathbin{\widehat{\wedge}} \sigma_2^{-1} \mathbin{\widehat{\wedge}} \sigma_1^{-1})$ *is equivalent to* $\sigma_1 \mathbin{\widehat{\wedge}} \sigma_1^{-1}$, *and, similarly,* $(\sigma_2 \mathbin{\widehat{\wedge}} \sigma_1^{-1} \mathbin{\widehat{\wedge}} \sigma_2^{-1}) \mathbin{\widehat{\vee}} (\sigma_2 \mathbin{\widehat{\wedge}} \sigma_2^{-1})$ *is equivalent to* $\sigma_2 \mathbin{\widehat{\wedge}} \sigma_2^{-1}$. *So*

$$\begin{array}{rcl} (\sigma_1 \gg \sigma_2) \mathbin{\widehat{\vee}} \sigma_3 & = & \sigma_1 \mathbin{\widehat{\vee}} \sigma_2 \mathbin{\widehat{\vee}} (((\sigma_1 \mathbin{\widehat{\wedge}} \sigma_1^{-1}) \mathbin{\widehat{\wedge}} (\sigma_2 \mathbin{\widehat{\wedge}} \sigma_2^{-1})) \mathbin{\widehat{\vee}} \sigma_3) \\ & = & \sigma_1 \mathbin{\widehat{\vee}} \sigma_2 \mathbin{\widehat{\vee}} ((\sigma_1 \mathbin{\widehat{\wedge}} \sigma_1^{-1} \mathbin{\widehat{\wedge}} \sigma_3) \mathbin{\widehat{\vee}} (\sigma_2 \mathbin{\widehat{\wedge}} \sigma_2^{-1} \mathbin{\widehat{\wedge}} \sigma_3)) \\ & = & \sigma_1 \mathbin{\widehat{\vee}} (\sigma_1 \mathbin{\widehat{\wedge}} \sigma_1^{-1} \mathbin{\widehat{\wedge}} \sigma_3) \mathbin{\widehat{\vee}} \sigma_2 \mathbin{\widehat{\vee}} (\sigma_2 \mathbin{\widehat{\wedge}} \sigma_2^{-1} \mathbin{\widehat{\wedge}} \sigma_3) \\ & = & (\sigma_1 \gg \sigma_3) \mathbin{\widehat{\vee}} (\sigma_2 \gg \sigma_3) \end{array}$$

In the following $(\sigma_1 \gg \sigma_3) \mathbin{\widehat{\vee}} (\sigma_2 \gg \sigma_3)$ will be used as an approximation for $(\sigma_1 \mathbin{\widehat{\vee}} \sigma_2) \gg \sigma_3$.

Example 11. Assume that Mark's desire for vegetarian meals is to take priority over his weight watching and his desire for hot food. This can be expressed as

$$\pi_{\text{ingredients}}\ \mathsf{is} \gg (\pi_{\text{spiciness}}\ \mathsf{best}\ \widehat{\vee}\ \pi_{\text{calories}}\ \mathsf{minimal}^{-1}).$$

This is equivalent to

$$\pi_{\text{ingredients}}\ \mathsf{is}\ \widehat{\vee}\ (\pi_{\text{ingredients}}\ \mathsf{is}\ \widehat{\wedge}\ \pi_{\text{ingredients}}\ \mathsf{is}^{-1}\ \widehat{\wedge}\ (\pi_{\text{spiciness}}\ \mathsf{best}\ \widehat{\vee}\ \pi_{\text{calories}}\ \mathsf{minimal}^{-1})),$$

the disjunctive normal form of which (with the maxima for each of the terms) is:

		$\sqcap$
	$\pi_{\text{ingredients}}\ \mathsf{is}$	vegetable biryani, pasta
$\vee$	$\pi_{\text{ingredients}}\ \mathsf{is} \wedge \pi_{\text{ingredients}}\ \mathsf{is}^{-1} \wedge \pi_{\text{spiciness}}\ \mathsf{best}$	vegetable biryani, chicken vindaloo, lamb casserole
$\vee$	$\pi_{\text{ingredients}}\ \mathsf{is} \wedge \pi_{\text{ingredients}}\ \mathsf{is}^{-1} \wedge \pi_{\text{calories}}\ \mathsf{minimal}^{-1}$	vegetable biryani, chicken vindaloo, lamb casserole

The intersection of these three maxima is {vegetable biryani} and this is the result that will be returned by this similarity measure.

Compare this with the maxima — {vegetable biryani, chicken vindaloo} — obtained by taking the conjunction rather than the prioritisation, as shown in example 6.

2.5.2 Preferences

An alternative approach is to first select maxima for the first similarity measure, and then take the maxima over these according to the second — i.e. the second similarity measure is applied only to discriminate between the maxima of the first. The *preference* of similarity measure σ_1 over similarity measure σ_2 is defined by:

Definition 7

$$\sqcap\ (\sigma_1 \rhd \sigma_2) = (\sqcap\ \sigma_2) \cdot (\sqcap\ \sigma_1)$$

The maxima of $((\sigma_1 \rhd \sigma_2)\ p)$ are the maxima $\sigma_1\ p\ S$, if there is no more than one maximum in $\sigma_1\ p\ S$, and the maxima $\sigma_2\ p\ S'$, where S' is the set of maxima $\sigma_1\ p\ S$, otherwise.

Example 12. Again assume that Mark's desire for vegetarian meals is to be preferred to his weight watching and his desire for hot food. Cases should then be selected by:

$$\sqcap\ (\pi_{\text{ingredients}}\ \mathsf{is} \rhd (\pi_{\text{spiciness}}\ \mathsf{best}\ \widehat{\vee}\ \pi_{\text{calories}}\ \mathsf{minimal}^{-1})),$$

which is equivalent to

$$(\sqcap\ (\pi_{\text{spiciness}}\ \mathsf{best}\ \widehat{\vee}\ \pi_{\text{calories}}\ \mathsf{minimal}^{-1})) \cdot (\sqcap\ \pi_{\text{ingredients}}\ \mathsf{is}).$$

The (unique) maximum found by applying this is again the vegetable biryani. Mark's preference for vegetarian meals will first select the vegetable biryani and the pasta as being better than the chicken vindaloo and the lamb casserole, and his second preference for either low calorie dishes or hot food will then select the vegetable biryani as being the better of these two.

2.5.3 Relating Priorities and Preferences

Priorities and preferences satisfy the following, for σ_1, σ_2 pre-order generating similarities:

Property 12

$$\begin{array}{rll} & (\sqcap\ \sigma_1)\ \widehat{\cap}\ (\sqcap\ \sigma_2)\ p\ S & \\ \subseteq & & (5) \\ & \sqcap\ (\sigma_1 \rhd \sigma_2)\ p\ S & \\ \subseteq & & (6) \\ & \sqcap\ (\sigma_1 \gg \sigma_2)\ p\ S & \\ \subseteq & & (7) \\ & \sqcap\ \sigma_1\ p\ S & \end{array}$$

Proof 12 *The proofs of the three inclusions in property 12 are:*

Inclusion (5)

$$\begin{array}{rl} & (\sqcap\ \sigma_1)\ \widehat{\cap}\ (\sqcap\ \sigma_2)\ p\ S \\ = & (\sqcap\ \sigma_1\ p\ S) \cap (\sqcap\ \sigma_2\ p\ S) \\ = & (\sqcap\ \sigma_1\ p\ S) \cap \{x \in S | \forall y \in S : x\ (\sigma_2\ p)\ y \Rightarrow y\ (\sigma_2\ p)\ x\} \\ = & \{x \in \sqcap\ \sigma_1\ p\ S | \forall y \in S : x\ (\sigma_2\ p)\ y \Rightarrow y\ (\sigma_2\ p)\ x\} \\ \subseteq & \{x \in \sqcap\ \sigma_1\ p\ S | \forall y \in \sqcap\ \sigma_1\ p\ S : x\ (\sigma_2\ p)\ y \Rightarrow y\ (\sigma_2\ p)\ x\} \\ = & \sqcap\ \sigma_2\ p\ (\sqcap\ \sigma_1\ p\ S)\ S \\ = & ((\sqcap\ \sigma_2) \cdot (\sqcap\ \sigma_1))\ p\ S \\ = & \sqcap\ (\sigma_1 \rhd \sigma_2)\ p\ S \end{array}$$

Inclusion (6) *First note that if σ_1 generates a preorder then*

$$x \in \sqcap\ \sigma_1\ p\ S \Rightarrow \{y \in S | x\ (\sigma_1\ p)\ y \wedge y\ (\sigma_1\ p)\ x\} \subseteq \sqcap\ \sigma_1\ p\ S \tag{8}$$

since:

$$\begin{array}{rl} & x \in \sqcap\ \sigma_1\ p\ S \wedge y \in \{y \in S | x\ (\sigma_1\ p)\ y \wedge y\ (\sigma_1\ p)\ x\} \\ \Leftrightarrow & (\forall z \in S : x\ (\sigma_1\ p)\ z \Rightarrow z\ (\sigma_1\ p)\ x) \wedge x\ (\sigma_1\ p)\ y \wedge y\ (\sigma_1\ p)\ x \\ \Rightarrow & \forall z \in S : y\ (\sigma_1\ p)\ x \wedge x\ (\sigma_1\ p)\ z \Rightarrow z\ (\sigma_1\ p)\ x \wedge x\ (\sigma_1\ p)\ y \\ \Rightarrow & \forall z \in S : y\ (\sigma_1\ p)\ z \Rightarrow z\ (\sigma_1\ p)\ y \\ \Leftrightarrow & y \in \sqcap\ \sigma_1\ p\ S. \end{array}$$

Now

$$\begin{array}{rl} & \sqcap\ (\sigma_1 \rhd \sigma_2)\ p\ S \\ = & \sqcap\ \sigma_2\ p\ (\sqcap\ \sigma_1\ p\ S) \\ = & \{x \in \sqcap\ \sigma_1\ p\ S | \forall y \in \sqcap\ \sigma_1\ p\ S : x\ (\sigma_2\ p\ S)\ y \Rightarrow y\ (\sigma_2\ p\ S)\ x\} \\ \textit{by (8)} & \\ \subseteq & \{x \in \sqcap\ \sigma_1\ p\ S | \forall y \in \{y \in S | x\ (\sigma_1\ p)\ y \wedge y\ (\sigma_1\ p)\ x\} : x\ (\sigma_2\ p)\ y \Rightarrow y\ (\sigma_2\ p)\ x\} \\ = & \{x \in \sqcap\ \sigma_1\ p\ S | \forall y \in S : (x\ (\sigma_1\ p)\ y \wedge y\ (\sigma_1\ p)\ x) \\ & \qquad \Rightarrow (x\ (\sigma_2\ p)\ y \Rightarrow y\ (\sigma_2\ p)\ x)\} \end{array}$$

Now, the expression above — $(x\ (\sigma_1\ p)\ y \wedge y\ (\sigma_1\ p)\ x) \Rightarrow (x\ (\sigma_2\ p)\ y \Rightarrow y\ (\sigma_2\ p)\ x)$ — is of the from $A \Rightarrow (B \Rightarrow C)$, with $A = x\ (\sigma_1\ p)\ y \wedge y\ (\sigma_1\ p)\ x$; $B = x\ (\sigma_2\ p)\ y$ and $C = y\ (\sigma_2\ p)\ x$, which, by simple

manipulation, is equivalent to $A \wedge B \Rightarrow A \wedge C$.

$$\begin{aligned} A \Rightarrow (B \Rightarrow C) &= \neg A \vee (B \Rightarrow C) \\ &= \neg A \vee \neg B \vee C \\ &= \neg A \vee \neg B \vee (A \wedge C) \\ &= \neg(A \wedge B) \vee (A \wedge C) \\ &= (A \wedge B) \Rightarrow (A \wedge C) \end{aligned}$$

Substituting this back in gives:

$$\begin{aligned} & \sqcap (\sigma_1 \rhd \sigma_2)\ p\ S \\ \subseteq\ & \{x \in (\sqcap\ \sigma_1\ p\ S) | \forall y \in S : x\ (\sigma_1\ p)\ y \wedge y\ (\sigma_1\ p)\ x \wedge x\ (\sigma_2\ p)\ y \\ & \qquad\qquad \Rightarrow \\ & \qquad\qquad x\ (\sigma_1\ p)\ y \wedge y\ (\sigma_1\ p)\ x \wedge y\ (\sigma_2\ p)\ x\} \\ =\ & \{x \in (\sqcap\ \sigma_1\ p\ S) | \forall y \in S : x\ ((\sigma_1 \mathbin{\hat{\wedge}} \sigma_1^{-1} \mathbin{\hat{\wedge}} \sigma_2)\ p)\ y \Rightarrow y\ ((\sigma_1 \mathbin{\hat{\wedge}} \sigma_1^{-1} \mathbin{\hat{\wedge}} \sigma_2)\ p)\ x\} \\ =\ & (\sqcap\ \sigma_1\ p\ S) \cap (\sqcap\ (\sigma_1 \mathbin{\hat{\wedge}} \sigma_1^{-1} \mathbin{\hat{\wedge}} \sigma_2)\ p\ S \\ =\ & \sqcap\ (\sigma_1 \mathbin{\hat{\vee}} (\sigma_1 \mathbin{\hat{\wedge}} \sigma_1^{-1} \mathbin{\hat{\wedge}} \sigma_2))\ p\ S \\ =\ & \sqcap\ (\sigma_1 \gg \sigma_2)\ p\ S \end{aligned}$$

Inclusion (7) *From definition 6*

$$\sqcap\ (\sigma_1 \gg \sigma_2)\ p\ S = \sqcap\ (\sigma_1 \mathbin{\hat{\vee}} (\sigma_1 \mathbin{\hat{\wedge}} \sigma_1^{-1} \mathbin{\hat{\wedge}} \sigma_2))\ p\ S,$$

and from property 10

$$\sqcap\ (\sigma_1 \mathbin{\hat{\vee}} (\sigma_1 \mathbin{\hat{\wedge}} \sigma_1^{-1} \mathbin{\hat{\wedge}} \sigma_2))\ p\ S = (\sqcap\ \sigma_1\ p\ S) \cap (\sqcap\ (\sigma_1 \mathbin{\hat{\wedge}} \sigma_1^{-1} \mathbin{\hat{\wedge}} \sigma_2)\ p\ S)$$

which is obviously a subset of $\sqcap\ \sigma_1\ p\ S$.

2.6 Conclusions

This section has presented a repertoire of basic (atomic) similarity measures, and a number of ways of combining these to form more complex measures. In the next section we will consider how this approach might be applied to the more traditional numeric valued similarity measures.

3 Cardinal Similarity Measures

Another type of similarity measure is one which returns a numeric value for a case, rather than a partial order on cases — i.e. rather than a similarity measure being a function $\sigma :: \Theta \to \Theta \to \Theta \to \mathbf{bool}$ it will be a function that "scores" cases $\sigma_{\mathbf{N}_\infty} :: \Theta \to \Theta \to \mathbf{N}_\infty$, or, alternatively, $\sigma_{[0,1]} :: \Theta \to \Theta \to [0,1]$ (in the following the first option will be taken). This scoring approach is less general than the one developed in section 2 because the orders defined by the scores are always total; it does not, for example, allow the possibility of incomparable cases. But some authors argue that such numeric measures have the advantage of giving cardinal as well as ordinal information to the user.

The definition of cardinal similarity measures again begins by considering comparison of individual elements of a case to seed values from a seed, and then considers how to combine the atomic measures, including the use of weightings.

3.1 Atomic Similarity Measures

The particular difficulty in defining numeric measures is how to treat elements of a case that have non-numeric types, e.g. how to score a case whose spiciness is mild, when the seed specifies a desired value of

hot. One approach is to order cases as in section 2.1 and then convert from the order to a numeric score. This is discussed in section 3.3, where it is shown that introducing the necessary cardinal information after ordering is problematic.

The alternative is to map non-numeric values to numeric ones. For example, spiciness values from mild to suicide might be mapped to $1, 2, 4, 8, 16, 32, 64, 128$ if, in our kitchen, a suicidal curry (128) has twice the ferocity of a killer curry (64), etc. In what follows, functions that carry out this mapping will be denoted as f. Obviously, for numeric valued attributes of cases, f will typically be the identity function, or, if the similarity measures are to return normalised values — i.e. some value in $[0, 1]$, rather than $\mathbf{N}_\infty$ — a normalising function.

Example 13. For the calories attribute the function f could be

- for similarity measures returning values in $\mathbf{N}_\infty$, the identity function;
- for normalising similarity measures, returning values in $[0, 1]$, the function

$$f\ n = 1 - 1/n$$

In most work on numerical measures [16, 7], equality of a value in a case to the seed is taken as a sign of similarity (corresponding to is below), or the difference between two values is taken as a sign of dissimilarity (corresponding to best below). But numeric correlates of flat, minimal and maximal can also be defined. In the definitions below, the higher the measure, the less similar the cases (some would call this a distance measure or dissimilarity measure).

$$\begin{array}{lll}
\textsf{flat:} & \textsf{flat}\ e\ x = \infty & \\
\textsf{is:} & \textsf{is}\ e\ x = 0, & \text{if } f\ x = f\ e \\
 & \quad = \infty, & \text{otherwise} \\
\textsf{minimal:} & \textsf{minimal}\ e\ x = f\ e - f\ x, & \text{if } f\ x \geq f\ e \\
 & \quad = \infty, & \text{otherwise} \\
\textsf{maximal:} & \textsf{maximal}\ e\ x = f\ e - f\ x, & \text{if } f\ x \leq f\ e \\
 & \quad = \infty, & \text{otherwise} \\
\textsf{best:} & \textsf{best}\ e\ x = |f\ x - f\ e| & \\
\textsf{id:} & \textsf{id}\ e\ x = f\ x &
\end{array}$$

An additional family of similarity measures can be defined which will be useful in discussing weightings in section 3.2. These measures will return a constant value for any case.

$$\textsf{const}_n\ e\ x = n$$

Clearly, flat is a special case of $\textsf{const}_n$, with $n = \infty$, and taking the inverse of a measure (subtracting, rather than adding the score given by that measure) is equivalent to multiplying by $\textsf{const}_{-1}$.

3.1.1 Other structures

It is also possible to derive cardinal similarity measures from the structures discussed in section 2.2 — trees, DAGs, graphs, user defined structures. All that is required is that a distance function (such as $\rightsquigarrow$ in section 2.2) be defined for these structures which can then be used to give the "score" for each case, rather than using the distance to define an ordering, as was done in section 2.2.

This could even be applied to numeric valued features by defining a distance function on numbers — e.g. a logarithmic distance function — and applying this directly. This provides an alternative to defining some function f and using operations such as best.

3.2 Combining Numeric Measures

Numeric similarity measures can be combined using basic arithmetic operations in a manner analogous to that presented in section 2.3.1. Using the operators $+$, $-$ (unary and binary), and $\times$, measures can be added, subtracted, multiplied and, using the $\textsf{const}_n$ measures and multiplication, weighted. Again, a normal form can be derived — a sum of products — and the products computed in parallel.

Example 14. Assume there is a numeric measure of spiciness. Assume also that Mark still prefers hot dishes, and this preference is 30 times[1] as important as his desire for a low calorie meal. Using the seed from example 1, with hot representing the spiciness value of a hot meal, the values of the cases in the case base can be retrieved by

$$(\pi_{\text{spiciness}}\ (\mathsf{const}_{30} \mathbin{\widehat{\times}} \mathsf{best})) \mathbin{\widehat{+}} (\pi_{\text{calories}}\ (\mathsf{const}_{-1} \mathbin{\widehat{\times}} \mathsf{minimal})),$$

the normal form of which is

$$(\pi_{\text{spiciness}}\ \mathsf{const}_{30}) \mathbin{\widehat{\times}} (\pi_{\text{spiciness}}\ \mathsf{best}) \mathbin{\widehat{+}} (\pi_{\text{calories}}\ \mathsf{const}_{-1}) \mathbin{\widehat{\times}} (\pi_{\text{calories}}\ \mathsf{minimal}).$$

The two terms in this sum can be computed in parallel, and the sum then taken. This is the equivalent for the cardinal case of the parallel evaluation of the maxima of the disjunctive normal form in the ordinal case.

Applying this similarity measure to the seed for Mark

$$\mathsf{mark} = (\text{“Mark”}, \text{False}, \mathsf{hot}\ (=16), 0)$$

and the case base given in example 1, and using the metric for spiciness given in section 3.1, gives the following values:

name	*spiciness*	*calories*	$\pi_{\text{spiciness}}\ \mathsf{const}_{30} \mathbin{\widehat{\times}} \pi_{\text{spiciness}}\ \mathsf{best}$	$\pi_{\text{calories}}\ \mathsf{const}_{-1} \mathbin{\widehat{\times}} \pi_{\text{calories}}\ \mathsf{minimal}$	“score”
lamb	mild (= 1)	634	450	634	1084
veg. biryani	medium hot (= 8)	843	240	843	1083
chicken vind.	extra hot (= 32)	634	480	634	1114
pasta	medium mild (= 2)	953	420	953	1373

Since the lowest score indicates the “best fit”, this measure will recommend the vegetable biryani, its closeness to the ideal spiciness hot (= 16) just compensating for its higher number of calories.

It is assumed here that f for the calories attribute is the identity function. The similarity measure given — $\pi_{\text{calories}}\ (\mathsf{const}_{-1} \mathbin{\widehat{\times}} \mathsf{minimal})$ — is chosen as being comparable to that used in the ordinal examples. It would, of course, be possible to simply use the id similarity measure.

3.3 Switching Types of Similarity Measure

It is fairly easy to switch from cardinal to ordinal similarity measures. To transform a cardinal measure to an ordinal measure the cardinal information can be used to generate an ordinal measure by comparing values. If $\sigma_{\mathbf{N}_\infty}$ is a cardinal similarity measure an ordinal measure can be defined:

$$\sigma\ p\ x\ y = (\sigma_{\mathbf{N}_\infty}\ p\ x) \geq (\sigma_{\mathbf{N}_\infty}\ p\ y),$$

The scores determine the ordering. Obviously the cardinal information — how much more highly one case scores than another — is lost in the transformation.

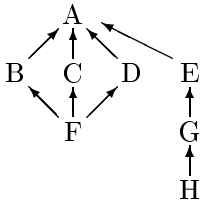
The transformation from an ordinal measure to a cardinal measure is more complex. The problem is in creating ordinal information where none was previously available, and in transforming a partial order to a total order. One possibility is to take the number of cases that can be found between two cases as an ordinal measure of the similarity of those two cases. Note that, as before, the higher the measure, the less similar the cases. However, if the cases are incomparable, there will be *no* objects between the two cases, and the measure will have to be adapted to deal with this. A possible transformation is, therefore

$$\begin{aligned} \sigma_{\mathbf{N}_\infty}\ p\ x &= \infty, \text{ if } |\{y | x\ (\sigma\ p)\ y\ (\sigma\ p)\ p \vee p\ (\sigma\ p)\ y\ (\sigma\ p)\ x\}| = 0 \\ &= |\{y | x\ (\sigma\ p)\ y\ (\sigma\ p)\ p \vee p\ (\sigma\ p)\ y\ (\sigma\ p)\ x\}|, \text{ otherwise} \end{aligned}$$

[1] Note that the choice of 30 is fairly arbitrary, and is partly driven by the expected relative size of the values returned by $\pi_{\text{spiciness}}$ best and π_{calories} minimal.

Note that if p and x are related, then the cardinality of the set of elements appearing between p and x will never be zero, since both p and x appear "between" p and x.

This measure can, however, give possibly counter-intuitive results. Consider the ordering, for seed A:



The cardinal measure proposed will return a higher (worse) value for F than for H, because there are five values between A and F (including A and F themselves), and only four between A and H. An alternative would be to take the shortest path between the elements, given by:

$$\begin{aligned}
\mathsf{minpath}\ x\ p &= \infty, \text{ if } \neg(x\ (\sigma\ p)\ \ p) \wedge \neg(p\ (\sigma\ p)\ x) \\
&= 0, \text{ if } x = p \\
&= 1 + \mathsf{min}\ \{\mathsf{minpath}\ x'\ p | x' \in \{y | x \oplus y \wedge x \neq y \wedge \nexists z \notin \{x,y\} : x \oplus z \oplus y\}\}, \text{ otherwise} \\
&\quad \text{where } \oplus = \sigma\ p, \text{ if } x\ (\sigma\ p)\ p \\
&\qquad\qquad\ = (\sigma\ p)^{-1}, \text{ otherwise}
\end{aligned}$$

3.4 Conclusions

It is certainly possible to apply the methods of section 2 to cardinal similarity measures. It is also possible to switch between the two types of measure. Note, however, that switching from a cardinal measure to an ordinal measure may entail some loss of information, and that switching from an ordinal measure to a cardinal one is not simple, and may produce counter intuitive results.

4 An Implementation

We have implemented a system that allows us to demonstrate some of the ideas that underly the parallel retrieval method for ordinal similarity measures. It should be stressed that the graphical demonstration of parallel retrieval is intended only as an expository aid, not a workable case based system.

The syntax of cases, seeds and similarity measures and the transformation of similarity measures to disjunctive normal form was defined using the ASF+SDF [8, 9] Meta-environment developed at the *Centrum voor Wiskunde en Informatica* and the University of Amsterdam. The disjunctive normal form generated by the system is then passed to Gadget Gofer [4, 13], a dialect of Gofer developed at the University of York for parallel programming of graphical user interfaces.

4.1 Installing a Case Base

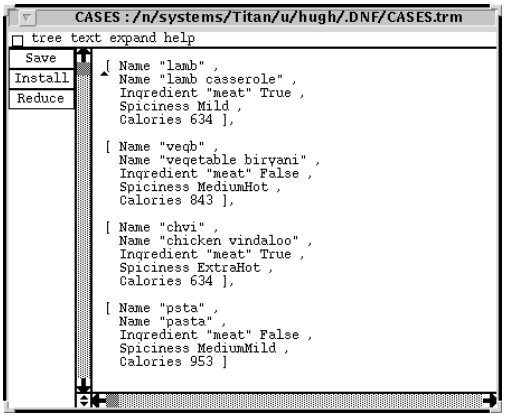
The system is called by `maxima` <*filename*>. A small window will pop up called "`Centaur menu`". Selecting "`ASF+SDF`" from the "`View`" pull-down menu will start the ASF+SDF system. The ASF+SDF system will automatically load the modules defining the case base language. The root window of the system will appear.

Syntax directed editors can be created by selecting the required module name from the "`Edit-term`" pull-down menu.

4.1.1 Case Memory.

At the moment the structure (type) of the case-base is "hard-wired" into the system. A possible future extension is to add a "meta-level", again using the ASF+SDF Meta-environment, which would allow the user to define their own case-base structure.

Selecting "`CASES`" from the pull-down menu will create a syntax directed editor (a "*term editor*") for a case memory. The case memory used in this example is:



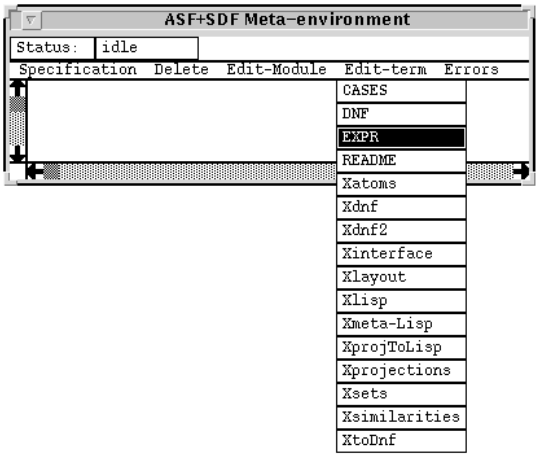
The cases here actually have two names, rather than one as in the first half of this report. The first name is a short version of the name which will be used to identify the case in the graphical demonstration of the system.

The case memory can be installed by first clicking on the "`Save`" button and then on the "`Install`" button.

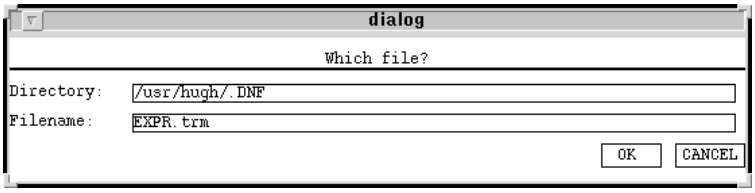
The use of the term editors will be discussed below in the presentation of the term editor for seeds and similarity measures.

4.1.2 Retrieval Request.

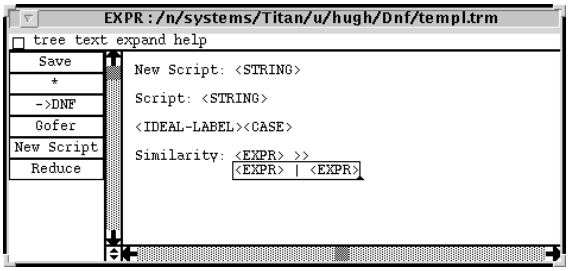
A seed and a similarity measure (together known as a "script") can be constructed by selecting "`EXPR`" in the pull-down "`EDIT-TERM`" menu.



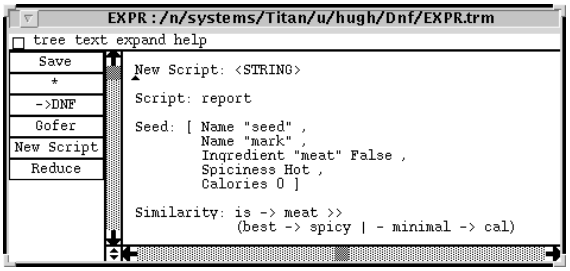
A pop-up box asks for confirmation of the choice.



Upon confirmation, a syntax directed editor (a "term editor") for a script will be created. At some point in the development it could look like this.



A final specification could be (the "meat" projection — `is->meat` — is a synonym for the "ingredients" projection — $\pi_{\text{ingredients}}$ `is` — used earlier in this report):



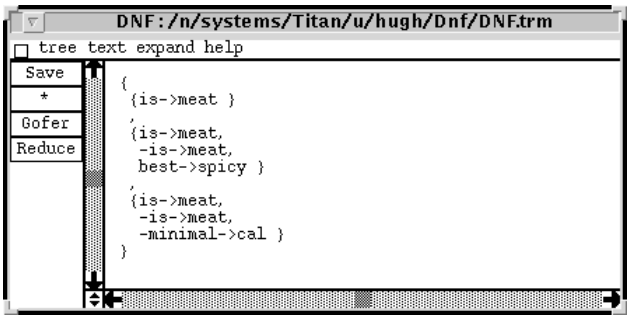
4.1.3 Other Options.

The other two user options in the "`EDIT-TERM`" menu are "`README`", which will pop up a window containing on-line documentation, and "`DNF`" which will pop up an editor for the most recent disjunctive normal form. The latter will usually only be used for viewing the normal form, although it may have some use in user-driven optimisation of the retrieval since the parallel retrieval mechanism can also be called from a disjunctive normal form term editor.

The remaining options (all prefixed with an 'X') are internal modules of the system.

4.2 Calling the Maximiser

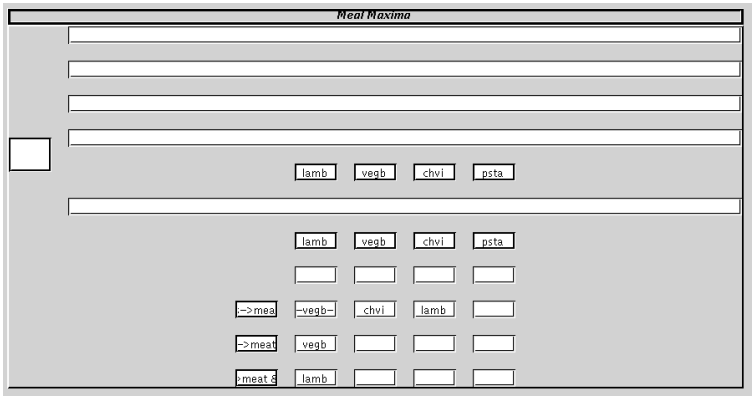
Once a script has been fully developed it can be installed by clicking on the "`Save`" button of the script term editor. If a case memory has also been installed the evaluation part of the system can now be called. This is done by clicking the "`*`" button of the "`EXPR`" term editor. The system will then compute the disjunctive normal form of the similarity measure provided (this will be available by way of the "`DNF`" option of the "`EDIT-TERM`" pull-down menu), and call Gadget Gofer. The disjunctive normal form of the example used here is (the minus sign indicates the inverse of the similarity measure):



The "`->DNF`" button of the script term editor will compute the disjunctive normal form without calling Gadget Gofer. Similarly, if the disjunctive normal form has already been computed the "`Gofer`" button will start the parallel retrieval without recomputing the disjunctive normal form.

4.3 Computing the Maxima

Gadget Gofer will create a window containing a demonstration of the parallel retrieval system. At an early stage of the computation the window will look like:



The working part of this window is the grid in the lower half. The top half of the window contains various explanatory sub-windows.

4.3.1 Computation

The working grid here consists of four labelled columns, each column corresponding to a case in the case memory; and four rows, one unlabelled and three labelled. The three labelled rows each correspond to a term in the disjunctive normal form being applied — here the normal form from example 11:

$$\begin{array}{ll} & \pi_{\text{meat}}\ \mathsf{is} \\ \widehat{\vee} & \pi_{\text{meat}}\ \mathsf{is} \mathbin{\widehat{\wedge}} \pi_{\text{meat}}\ \mathsf{is}^{-1} \mathbin{\widehat{\wedge}} \pi_{\text{spiciness}}\ \mathsf{best} \\ \widehat{\vee} & \pi_{\text{meat}}\ \mathsf{is} \mathbin{\widehat{\wedge}} \pi_{\text{meat}}\ \mathsf{is}^{-1} \mathbin{\widehat{\wedge}} \pi_{\text{calories}}\ \mathsf{minimal}^{-1}. \end{array}$$

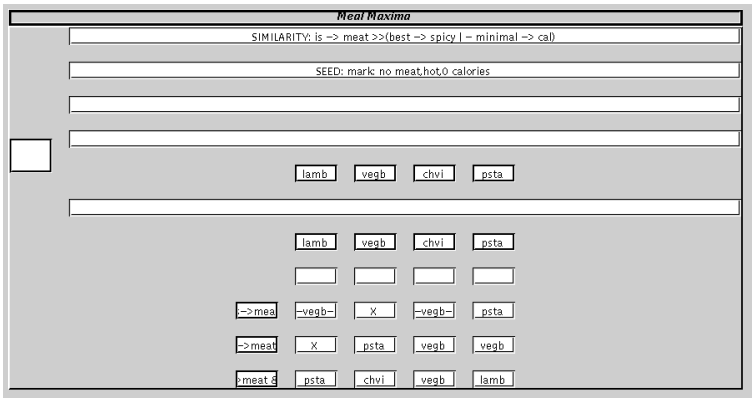
The final result of the computation will appear in the unlabelled row.

The lower 3×4 right hand corner of the grid shows the current state of the computation. Each column corresponds to a case in the case base (indicated by the short version of its name) and each row to a term in the disjunctive normal form being applied. More information about both terms and case base members can be obtained in the explanatory windows (see below).

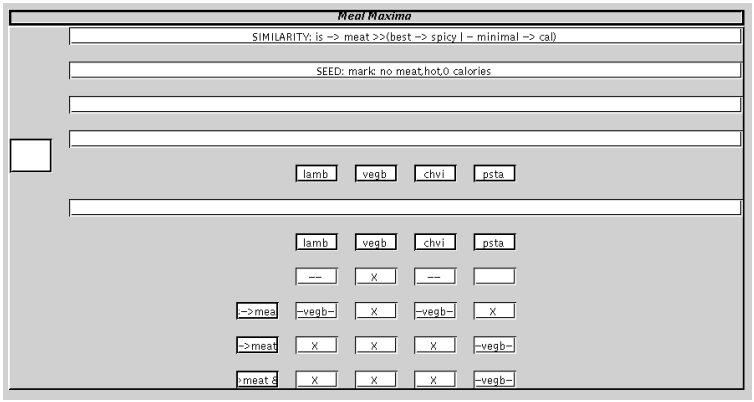
Cases in the case base are fed into each row from the left, in the same order as the rows when reading from left to right. These are then compared to the case belonging to the column. Each window shows the

case currently being compared with the column's case. So, for example, at the point in the computation shown above, in the top row-second column window "`chvi`" indicates that the vegetable biryani has already been compared to the lamb caserole and itself, and is currently being compared to the chicken vindaloo. If a column's case is found to be worse than the case currently being considered — i.e. if $\neg(c \oplus^t r \Rightarrow r \oplus^t c)$, where c is the fixed case for the column, r is the current case, and $\oplus^t$ is the term for the row in question — it, the case for this column, is eliminated in this row. This is indicated by placing hyphens around the name of the case that eliminated it. For example, above, in the top row, the lamb casserole has been found to be worse than the vegetable biryani, and has been eliminated.

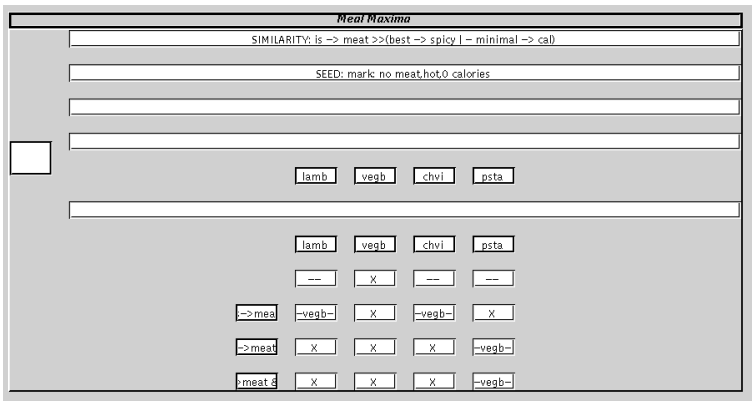
If a case has not been eliminated after being compared with all cases it is a maximum for this term. This is indicated by placing an "`X`" in its window. This can be seen in the following snapshot.



Here the vegetable biryani has been found to be at least as good as all the other cases for the first term, and is therefore a maximum for this term. Similarly the lamb casserole is a maximum for the second term. Computation is complete for all but the last case in the first term, but is still proceeding for all other cases. As computation is completed in each column the intersection can be taken. The result of this is shown in the remaining row of the grid — the row that has, until now, remained blank.

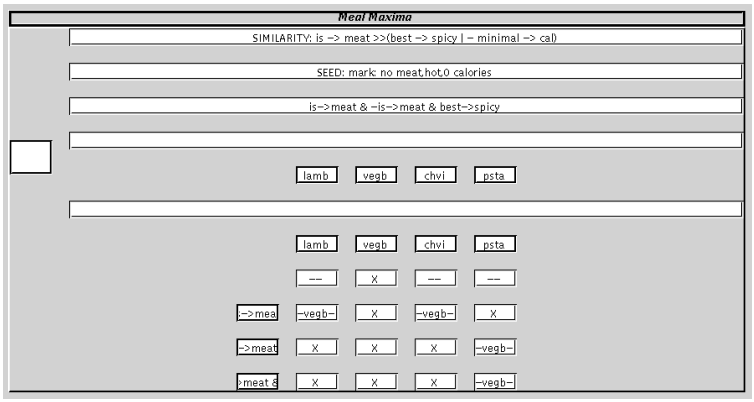


The lamb casserole is not maximal (since it was eliminated by the vegetable biryani under the first term), and neither is the chicken vindaloo (also having been eliminated by the vegetable biryani under the first term). The vegetable biryani is maximal. The result for the pasta has not yet been finalised. The final result is given in the next snapshot.

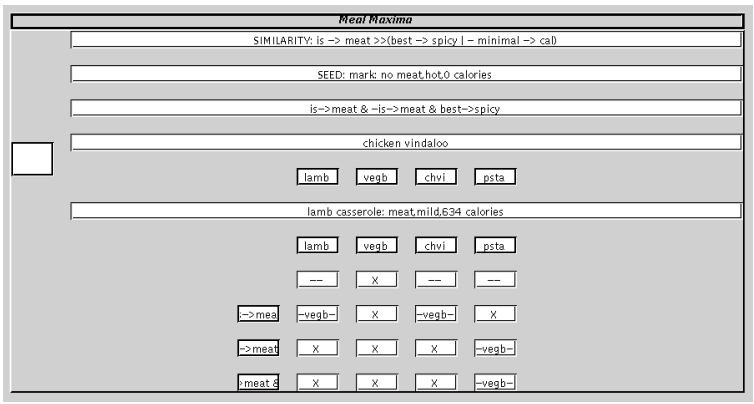


4.4 Explanations

The rest of the window is devoted to explanatory systems. The topmost window shows the similarity measure being applied, the next window down the seed to which it is applied. The remaining three windows are for presenting information about terms and cases. Clicking on a button to the left of one of the named rows will cause the corresponding term to be displayed in the third explanatory window. The term in the second row is, for example, π_{meat} $\mathsf{is}\ \widehat{\wedge}\ \pi_{\text{meat}}$ $\mathsf{is}^{-1}\ \widehat{\wedge}\ \pi_{\text{spiciness}}$ best, as shown here.



The last two explanatory windows are for cases. These can be activated by clicking on the buttons immediately below them. Clicking once will give the full name of the case (as illustrated below for "`chvi`"), clicking twice the full specification (shown here for "`lamb`").



5 Conclusions

We have presented a repertoire of tools for constructing similarity measures, both numeric and symbolic. These tools make it possible to construct similarity measures systematically and/or incrementally, in which a more refined similarity measure is derived from the result of applying a simpler measure.

The implied loss of efficiency that the use of more flexible similarity measures entails can be compensated for by the opportunities this method offers for parallel evaluation.

A system has been developed that demonstrates the ideas presented. Currently this provides a graphical demonstration of the method. Work is in progress to develop this into a realistic, efficient system, and to extend the work to cover other knowledge manipulation systems [14].

References

[1] A. Aamodt and E. Plaza. Case based reasoning: Foundational issues, methodological variations and system approaches. *AI Communications*, 7(1):39–59, 1994.

[2] R. Bareiss, J.A. King, J. Ashley, K. Kolodner, B. Porter, and P. Thagard. Panel on "similarity metrics". In *Proceedings of DARPA Case Based Reasoning Workshop*, pages 66–84. Morgan Kaufmann, 1989.

[3] M. Brown. *A Memory Model for Case Retrieval by Activation Passing.* PhD thesis, Department of Computer Science, University of Manchester, 1994. Technical Report 94-2-1.

[4] Functional Programming Group, Department of Computer Science, University of York, UK. *Gadget Gofer On-Line Reference Manual.* `http://dcpu1.cs.york.ac.uk:6666/rjn/ggref/index.html`.

[5] A.D. Griffiths and D.G. Bridge. Formalising the knowledge content of case memory systems. In Ian D. Watson, editor, *Progress in Case-Based Reasoning:* First United Kingdom Workshop in Case-Based Reasoning, pages 32–41. Springer Verlag *Lecture Notes in Computer Science; 1020; Lecture Notes in Artificial Intelligence*, 1995.

[6] D. Hennessy and D. Hinkle. Applying case-based reasoning to autoclave loading. *IEEE Expert*, 7(5):21–26, 1992.

[7] P. Klahr and G. Vrooman. Commercialising case based reasoning technology. In I.M. Graham and R.W. Milne, editors, *Research and Development in Expert Systems VIII*, pages 18–24. Cmabridge University Press, 1991.

[8] P. Klint. *The ASF+SDF Meta-Environment User's Guide, version 26.* Centrum voor Wiskunde en Informatica (CWI), February 1993. Available by *ftp* from `ftp.cwi.nl:pub/gipe/reports` as `SysManual.ps.Z`.

[9] P. Klint. A meta-environment for generating programming environments. *ACM Transactions on Software Engineering and Methodology*, 2(2):176–201, 1993.

[10] J.L. Kolodner. Judging which is the "best" case for a case-based reasoner, 1989. published as part of [2], pages 77–81.

[11] J.L. Kolodner. *Case Based Reasoning.* Morgan Kaufmann, 1993.

[12] P. Myllymäki and H. Tirri. Massively parallel case-based reasoning with probabilistic similarity measures. In S. Wess, K.D. Althoff, and M.M. Richter, editors, *Proceedings of the 1st European Workshop on Case-Based Reasoning, Lecture Notes in Computer Science No. 837*, pages 144–154. Springer Verlag, 1994.

[13] Rob Noble and Colin Runciman. Lazy functional components for graphical user interfaces. In *Proceedings of PLILP*, 1995.

[14] Hugh Osborne and Derek Bridge. Relational metrics: Integrating case based reasoning and database management systems. Technical report, Department of Computer Science, University of York, 1997. In preparation.

[15] B.W. Porter. Similarity assessment: Computation vs. representation. In *Proceedings of DARPA Case Based Reasoning Workshop*. Morgan Kaufmann, 1989.

[16] M.M. Richter and S. Wess. Similarity, uncertainty and case-based reasoning in PATDEX. In R.S. Boyer, editor, *Automated Reasoning: Essays in Honour of Noody Bledsoe*, pages 249–265. Kluwer, 1991.

[17] C.K. Riesbeck and R.C. Schank. *Inside Case Based Reasoning*. Lawrence Erlbaum Associates, 1989.

[18] M. Ryan. Prioritising preference relations. In S.J.G. Burn and M.D. Ryan, editors, *Theory and Formal Methods 1993: Proc. Imperial College Department of Computer Science Workshop on Theory and Formal Methods*. Springer Verlag, 1993.

[19] E. Simoudis. Using case-based retrieval for customer technical support. *IEEE Expert*, 7(5):7–13, 1992.