Integrating Order and Distance Relationships from Heterogeneous Maps

Mark Graves
Artificial Intelligence Laboratory
University of Michigan
1101 Beal Avenue
Ann Arbor, MI 48109-2110
graves@engin.umich.edu

Abstract

There is no automatic mechanism to integrate information between heterogeneous genome maps. Currently, integration is a difficult, manual process. We have developed a process for knowledge base design, and we use this to integrate order and distance relationships between genetic linkage, radiation hybrid, and physical maps.

Until now, the only way to develop a persistent, knowledge-intensive application was to either develop a new knowledge base from scratch or coerce the application to fit an existing knowledge base. This was not from lack of interest by the knowledge base or database community, but merely from a lack of theoretical tools powerful enough to tackle the problem. We import formalisms from knowledge representation, natural language semantics, programming language research, and databases. These form a strong, theoretical foundation for knowledge base design upon which we have implemented the knowledge base design tool called WEAVE.

Introduction

What does a geneticist want from a database?

One thing is a database which integrates the different kinds of genome maps.

What would you ask the database?

What is the distance between markers? Is there support for one order over another? What is the consistency between marker orders?

We have implemented a knowledge base design tool which makes it easier to develop knowledge bases which answer these kinds of queries.

We have developed knowledge bases within several other areas including general knowledge representation, problem solving, and natural language processing, with positive results, but have found the heterogeneous genome mapping problem to be the most in need of direct knowledge base support. Currently, there are many different kinds of genome maps at different levels of granularity, with different properties, and with different ways in which they are useful. Each map is based on laboratory procedures which can have errors and inconsistencies. Different statistical methods are used to deal with the problems, and they are based on different assumptions and models. People can generally deal with one kind of map at a time, though it is tedious. When multiple, heterogeneous maps are available, it can be difficult to handle the complexity.

We have developed a general process for designing knowledge bases which can be used for heterogeneous genome maps. This process is supported by a strong, theoretical foundation and an implemented knowledge base design tool called WEAVE [Graves, 1993]. We demonstrate the process on a simple representation for distance and explain how queries can be asked of the knowledge base. We also show how order information can be represented in a similar fashion.

Knowledge Base Design

Often the best way to solve a problem is to change the way the problem is viewed [Anderson, 1985]. This requires a change in the representation of the problem state. However, most representation schemes require that a problem be represented in only one way. This can lead to a more efficient implementation, but requires a human to mentally coerce their reasoning process into a fixed, unnatural form (while trying to solve a difficult problem).

The solution to this problem is to have one formalism to represent the structure of the knowledge in a computationally effective form and let the user view the data in the manner most natural to the solution of the problem. If the user is unsure of the most natural representation, it is also important that the system be both flexible and extensible.

We have applied this to the problem of knowledge base design and have implemented a tool which can be used to develop knowledge bases that allow for multiple views of the same structure. This is done by allowing distinct data types to share a common structure for the data and is implemented via the layered architecture of WEAVE.

Integrating heterogeneous maps is an especially good

problem on which to demonstrate this approach because there is already an underlying structure (the genome) which people view in different ways (physical and genetic maps). This is not to say that the most computationally efficient way of representing the underlying structure of the maps will correspond to the genome, but merely indicates that there is a common structure to the maps, and this can guide development toward a more effective implementation. It gives us a place to start and fixes the user's view to be the heterogeneous maps. This results in the goal to find a common structure which can be efficiently used to integrate the information contained in multiple, heterogeneous maps.

To design an application-specific data model using our approach, the knowledge base developer begins with a graphical sketch which appears to capture the structure and semantics required for the application. The developer abstracts common features of the sketch and uses WEAVE to group the abstractions into new data types. Methods are then developed to do reasoning on the data types, and the types and methods are collected to form a new data model for the knowledge base. Any step can be repeated to refine the data model, and WEAVE is used to develop a prototype knowledge base. We demonstrate this process on the problem of representing distance and order information for heterogeneous genome maps.

Genome Mapping

Mapping is the process of determining the relative position of genes and other genetic markers on a chromosome and estimating the distance between them. Markers have a physical location on a chromosome which can be identified by some laboratory procedure and whose pattern of inheritance can be followed. A genome map can be used to find the location of a specific gene whose location is not known by using laboratory procedures to discover which markers on the map are close to the gene in question. There are several different mapping processes and strategies with several resulting maps. In this paper, we deal with three different maps: genetic linkage maps, physical maps, and radiation hybrid maps.

Genetic linkage maps are based on the inheritance of genes and markers from one generation to another [Ott, 1991]. Alternative forms of a marker (alleles) are studied within a pedigree (family) to determine their pattern of inheritance. Multiple markers can be examined, and statistical methods can be used to estimate the likelihood that they are linked, that is, close together on the same chromosome. Distance can be measured as the expected number of recombination events (crossovers) which occur between markers. This distance is measured in Morgans with 1 Morgan corresponding to one expected crossover per meiosis.

Physical maps vary in their degree of resolution depending upon the laboratory procedure used. They measure physical distance between markers in terms of the number of base pairs between two markers; the actual distance can only be estimated based on the resolution of the specific laboratory procedure used.

Radiation hybrid maps are created by using a high dose of x-rays to break a human chromosome into several fragments. Laboratory procedures can be used to collect fragments into rodent-human hybrid clones which are analyzed for the presence or absence of specific markers. Each hybrid contains a sample of human fragments and statistical methods can be used to estimate the probability of a radiation-induced break between two markers. It appears that the frequency of breakage between markers is directly proportional to physical distance, and this distance can be recovered using statistical methods which take the possibility of multiple, intervening breakpoints into account [Cox *et al.*, 1990]. The distance is measured in Rays with 1 Ray corresponding to one expected break. Radiation hybrid maps attempt to measure physical distance (as do physical maps), but do this by breaking the chromosome at random locations, which requires statistical methods to recover distance (as is needed for genetic maps).

Theoretical Foundations

Until now, the only way to develop a persistent, knowledge-intensive application was to either develop a new knowledge base from scratch or coerce the application to fit an existing knowledge base. This was not from lack of interest by the knowledge base or database community [Brodie, 1984, Carey, 1990, Zdonik and Maier, 1990], but merely from a lack of theoretical tools powerful enough to tackle the problem.

The knowledge base design process can deteriorate into *ad hoc* development unless there are theories supporting it which are both powerful enough to express the needed information and natural enough to guide the development in a productive manner. One good way of representing structural information is a flexible, graphical framework. Behavioral information can be stored in a strongly-typed system (as is done in the functional and object-oriented programming language paradigms). Static and dynamic properties of knowledge-intensive applications are traditionally organized in terms of a data model.

We import attribute value description languages [Nebel and Smolka, 1990] from knowledge representation [Brachman and Schmolze, 1985][Aït-Kaci and Podelski, 1991] and natural language semantics [Kasper and Rounds, 1986, Sowa, 1984] to represent the structure of knowledge in a graphical framework, and we import constructive type theory [Martin-Löf, 1982] from programming language research to represent type information in a manner that lends itself to the automatic generation of inference rules on the type. These, together with

a persistent, vivid, binary logic knowledge store [Etherington *et al.*, 1989, Deliyanni and Kowalski, 1979, Bic and Lee, 1987] and an algebraic approach to data models, form a strong, theoretical foundation for knowledge base design upon which we have implemented the tool WEAVE.

Attribute value description languages [Nebel and Smolka, 1990] represent information in a framework of concepts and attributes which has a Tarski-style, model-theoretic semantics but can represent the information usually found in graphical notations. These have been used for a variety of applications, and we have developed an attribute value description language within WEAVE called WEB which is geared toward knowledge base design.

Constructive (intuitionistic) mathematics is a non-classical approach which does not allow for indirect proofs. Constructive type theory encodes logical propositions as types in a formalism which allows mathematical proofs to be tightly coupled to computer programs. It uses natural deduction style inference rules to develop and reason with types in a manner which is both mathematically rigorous and computationally perspicuous. Constructive type theory is a theory of types which defines type constructors and data constructors in terms of inference rules, and abstract data types are created by type constructors. For example, LIST(A), where A is a type variable, is a type constructor which creates lists; LIST(SYMBOL) is an abstract data type for list of symbols whose elements are created by the data constructors *pair* and *nil*.

Data models are organized using an algebraic approach and are built up using the constructive type definitions and the operations are developed as methods on the types.

Knowledge Base Querying

One advantage of a knowledge base over an *ad hoc* system is the ability to query against it. Because we want the knowledge base to be useful in a realistic setting, it is also important to make the interface as user-friendly as possible. Query processing is done in WEAVE through a simple knowledge base manager. Currently, the knowledge base manager is given a partially instantiated data constructor and retrieves the structures in the knowledge base which match it.

Although this is work in progress, we want to set the context in which knowledge base design is most useful. We are developing a natural language interface to the knowledge base manager which will allow for English queries to the knowledge base such as:

Find the distance between marker D21S1 and marker D21S11.
Find the best orderings.
Find order evidence for markers D21S16 and D21S48.

WEAVE is being used to implement the natural language interface, and this natural language interface application will also serve as another test and demonstration of WEAVE's effectiveness.

WEAVE can answer these queries and others like them now when expressed as data constructors such as:

```
distance(marker('D21S1), marker('D21S11), ?x)
?order in best-order(?dataset, ?order)
order(marker('D21S16), marker('D21S48), ?order)
```

The natural language queries and data constructor queries have a similar form which can be used in a unification-based natural language interface [Shieber, 1986]. The disadvantage in all implemented systems except ours is that this restricts the queries to have a form similar to the data constructors that were used to define the knowledge base.

In WEAVE it is possible to have multiple, overlapping type definitions on the same structure. This allows data to be entered using one view of the structure and retrieved using alternative views.

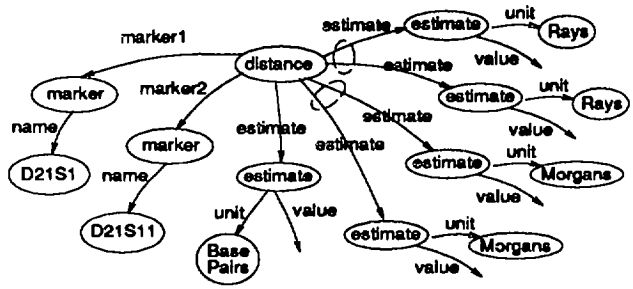
We now show how one view can be created for DISTANCE. This is demonstrated in terms of putting data into the knowledge base, although the same types are also used for retrieval. The same process is also used to develop overlapping types for retrieving the data.

Distance

Distance between markers in a genome map represents: expected number of recombination events (crossovers) between them, expected number of breaks induced by irradiation, or physical distance expressed in base pairs. Each of these distances can be estimated by a laboratory procedure. We give a common representation for the distances and their estimates, develop data types for them, and show how they can be combined to integrate distances from heterogeneous maps.

Abstracting Common Features

Distance between markers in a map can be represented graphically as a *distance* node with estimates of the distance represented as values of a *multi-valued attribute* (set-valued role) labeled *estimate*. For example, the distance between the markers D21S1 and D21S11 may be represented as:



where the estimates are defined by multiple data sets. These estimates should be thought of as being collected by the *units* of the distance estimates, and the collections are denoted above by the dashed lines.

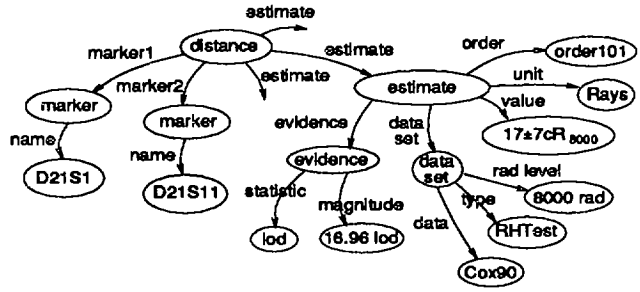
From this sketch an abstraction for distance can be formed in the binary logic programming language WEB which will construct the graph for distance.

wDistance(*?marker1*, *?marker2*, *?estimate*)
≡ [create *?distance*]
(*marker1* *?distance* *?marker1*)
(*marker2* *?distance* *?marker2*)
(*estimate* *?distance* *?estimate*)
[return *?distance*]

where *?name* denotes a variable, and we use *w*⟨*Constructor*⟩ to make clear that this is a program defined in WEB. The abstractions in WEB that construct graphs in the knowledge base are called *graph constructors.*

The graph constructor *wDistance* is then associated with a data constructor for the data type DISTANCE. The data constructors are embedded in a functional programming language and are used to build the knowledge base.

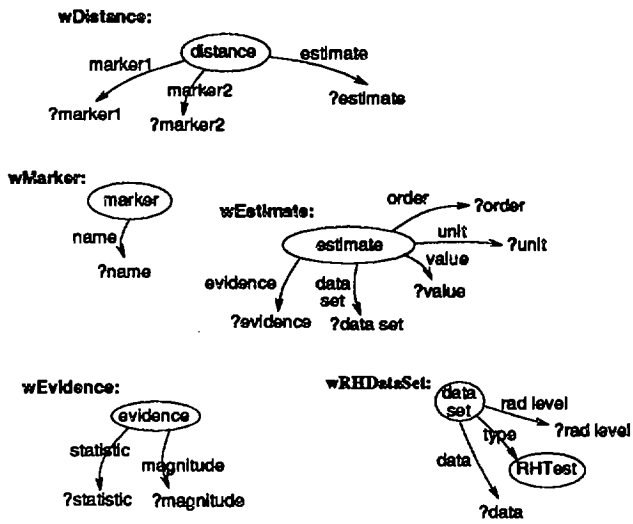
Much more information is needed in a representation of the estimate, such as the data set used, the order which the distance is based on, and the statistical evidence for the estimate. When these are included, it results in a representation such as:



which represents the distance information between D21S1 and D21S11 from a radiation hybrid data set [Cox *et al.*, 1990].

Abstractions, data constructors, and data types can be generated from this sketch as follows. First, find the sections of the graph which are likely to be reused in a semantically meaningful manner. In this example, the concepts involved are: distance, marker, estimate, evidence, and data set. Each of these concepts are associated with a section of the graph. We then define a graph constructor to build each section (as was done for *distance* above). For this example, it is fairly straightforward to do, though WEAVE also has more extensive capabilities which can also deal with more complex constructs, such as cyclic graphs, collections of multi-valued attributes, and indirection.

When we separate the sections of the graph, we are left with five graph constructors: *wDistance*, *wMarker*, *wEstimate*, *wEvidence*, and *wRHDataSet* which build the graphs



These five graph constructors have arguments as follows:

wDistance(*?marker1*, *?marker2*, *?estimate*)
wMarker(*?name*)
wEstimate(*?value*, *?unit*, *?dataset*, *?evidence*, *?order*)
wEvidence(*?statistic*, *?magnitude*)
wRHDataSet(*?radlevel*, *?data*)

Note that in this example the *value* includes both the estimate and a measure of variability.

Forming Data Types

Each of these graph constructors is associated with a data constructor for a user-defined type. The data constructor's parameters are typed and accessed through a strongly-typed functional programming language called SPIDER (also part of WEAVE). The graph is created when the data constructor is evaluated within SPIDER. The data constructors have type specifications like:

distance: *Marker* × *Marker* × *Estimate* → *Distance*
marker: *Symbol* → *Marker*
estimate:
Number × *Unit* × *DataSet* × *Evidence* × *Order* → *Estimate*

where the types are created using a **defstype** form in SPIDER. The **defstype** form for DISTANCE is:

```
defstype Distance ()
  distance(Marker, Marker, Estimate) = wDistance
```

Most types have more than one data constructor. The common data type LIST has data constructors `nil()` and `pair(?x,?l)`, and the type BINARYTREE has data constructors `leaf(?a)` and `node(?left,?right)`. For the current example, we have found it useful to have multiple data constructors for DATASET — for both a radiation hybrid data set, which takes the radiation level as another argument.

and data sets that do not need that additional argument. Thus for DATASET, there are data constructors **wDataSet** and **wRHDataSet** with a type definition of

```
defstype DataSet (A)
  dataset(DataSetType,A) = wDataSet
  rad-dataset(Number,A) = wRHDataSet
```

The DATASETTYPE would be either *Genetic* or *Physical* in this example, as is shown in the next section. These data constructors can be used to build the knowledge base. The graph above can be built by the expression

```
distance(marker('D21S1),
         marker('D21S11),
       estimate(0.17, Rays,
                rad-dataset(8000, 'Cox90),
                evidence(lod, 16.96),
                order(...) -- explained below
                ))
```

Functions are defined on the data types, and their execution is specified by a collection of inference rules in constructive type theory. These functions correspond to methods in object-oriented programming. These inference rules have traditionally been used for type inference or automated reasoning [Backhouse *et al.*, 1988, Constable *et al.*, 1986], but we use them to give an operational semantics to functions which operate on elements of the type. Some inference rules tell how to form the type and data constructors. For DISTANCE, these look like

Distance-formation

Distance type

distance-introduction

$$\frac{m1 \in \textit{Distance} \quad m2 \in \textit{Distance} \quad e \in \textit{Estimate}}{\textit{distance}(m1, m2, e) \in \textit{Distance}}$$

These inference rules are calculated in a straightforward manner from the **defstype** form above. We also have developed an algorithm as part of WEAVE which will calculate inference rules that tell how to eliminate a type into its constituents and perform computations on the type. These rules are then used to create a form in SPIDER which performs well-founded computations, i.e., under certain liberal conditions the computation can be guaranteed to halt.[1] If this proves overly restrictive for some application, a full (recursively enumerable), functional programming language, such as Lisp or SML [Milner *et al.*, 1990], is also available for the cases where it is necessary.

The elimination and computation rules for DISTANCE are somewhat more complex and are omitted. The details of the rules are not important for the understanding of how they are used. The elimination rule describes a function called **Distance-elim** which takes three arguments. The first is the expression to be evaluated. The others are lambda expressions that give the body of the function which is to be applied to the expression depending upon what the outermost data constructor of the expression is. The computation rules tell how the second and third arguments to **Distance-elim** are to be used to calculate the result. This is translated into a lambda expression which is then evaluated using lazy evaluation when applied to a *distance*.

For example, a function to collect all the estimates of a distance into a list, regardless of the data set or units, could be defined by the user in SPIDER as:

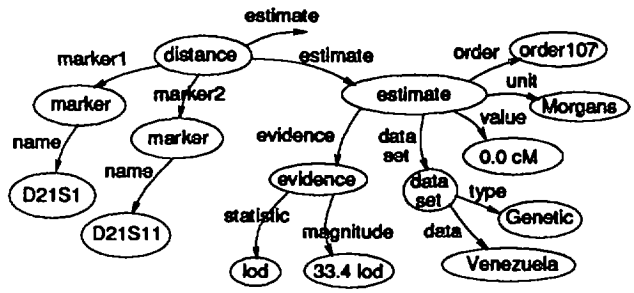
```
defsfun values (Distance) ()
  distance(?m1,?m2,empty) => nil()
  distance(?m1,?m2,?e)::?next
    => pair(estimate-value(?e),
            recurse(?next))
```

This is translated into the lambda expression:

$\lambda x.\textit{Distance-elim}(x, \textit{nil}(),$
$\quad \lambda m1.\lambda m2.\lambda e.\lambda r.\lambda i.\textit{pair}(\textit{estimate-value}(e), i))$

Results for Heterogeneous Mapping

Although the process of defining a DISTANCE type was given for radiation hybrid mapping, a similar process can be used for other maps. The DISTANCE type can be used for genetic maps, and distance information can be shared between heterogeneous maps. For example, consider a representation of the distance between D21S1 and D21S11 from a genetic map taken from [Tanzi *et al.*, 1988]:



The same data constructors can be used in this instance as were used above for radiation hybrid distances.

```
D21S1 == marker('D21S1)
D21S11 == marker('D21S11)
VenDataSet == dataset(Genetic, 'Venezuela)
distance(D21S1,  D21S11,
       estimate(0.0, Morgans, VenDataSet,
                evidence(lod, 33.4),
                order(...) -- explained below
                ))
```

[1] This occurs because all elements of a type must have been constructed through a finite (though unlimited) application of introduction inference rules. In addition, the functions on the type must be restricted to the primitive recursive functions.

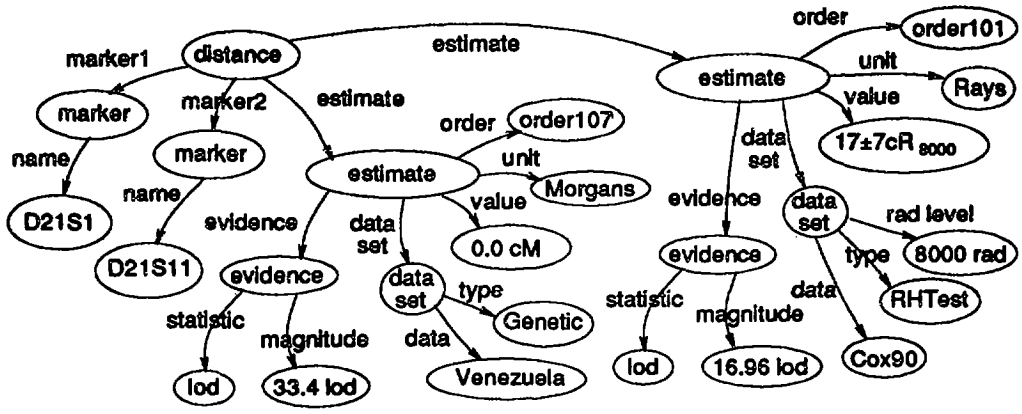


Figure 1: Distance information between D21S1 and D21S11 from a radiation hybrid and genetic map.

This leads automatically to a combined graphical representation in the knowledge base as shown in Figure 1.

Distance information from additional maps can also be added, and queries asking for specific information can be asked of the knowledge base.

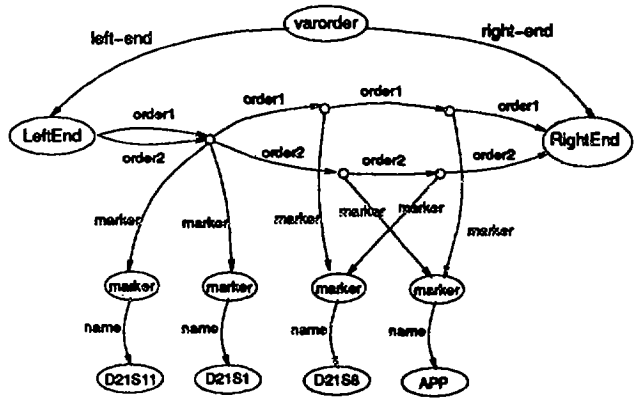
Order

Results similar to the types for DISTANCE can be obtained for order information. There are three dimensions of an order representation that we deal with here: intra-order uncertainty, inter-order uncertainty, and heterogeneity of maps. Intra-order uncertainty occurs when no order information is available for a collection of markers: they are either physically indistinguishable or tightly linked with no intervening crossovers or radiation breaks observed. Inter-order uncertainty occurs when markers can be distinguished, but there is still some uncertainty as to which is the actual order within the collection of markers and/or with respect to other markers, though one order may be more likely than another. Map order information can come from heterogeneous maps with different levels of granularity and sometimes conflicting orders.

To deal with this information we must represent:

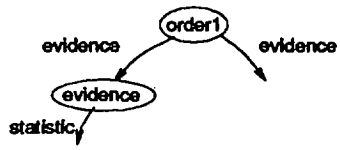
1. the order of markers and sets of markers,
2. a collection of orders which may be "partially ordered" by some likelihood statistic, and
3. collections of orders which may overlap in the markers ordered, but may conflict and have omitted data.

Although it is useful to have a simple way to represent known order for a collection of markers, it appears, for the general case, a representation is needed such as:



which represents uncertainty in the order D21S11, D21S1, D21S8, and APP from a genetic linkage map [Warren *et al.*, 1989]. The markers D21S11 and D21S1 are tightly linked with no observed crossovers, and the order D21S11/D21S1 – D21S8 – APP is only 235 times more likely than D21S11/D21S1 – APP – D21S8 (usually not considered statistically significant because it is less than 10^3).

Abstractions, data constructors, and data types are then formed in a manner similar to the process for DISTANCE. In addition, the arcs in the graph, e.g., order1 and order2, can also be treated as nodes (thus a higher-order, binary logic programming language). This allows auxiliary information, such as statistical evidence, to be associated with an order. This is represented as:



and abstractions can be formed in like manner.

Order information from multiple, heterogeneous maps can be combined using the same data types. For example, we can enter order information from a phys-

ical map [Gardiner and Patterson, 1992], a radiation hybrid map [Cox *et al.*, 1990], and two genetic maps [Warren *et al.*, 1989, Tanzi *et al.*, 1992] of a portion of human chromosome 21. Each map is entered separately, but because of the overlap in markers, a knowledge base which includes the information in Figure 2 is the result.

This represents the most likely order from each map. In this case, there are no conflicting orders, and a potential overall order can be obtained from WEAVE as is shown in Table 1.

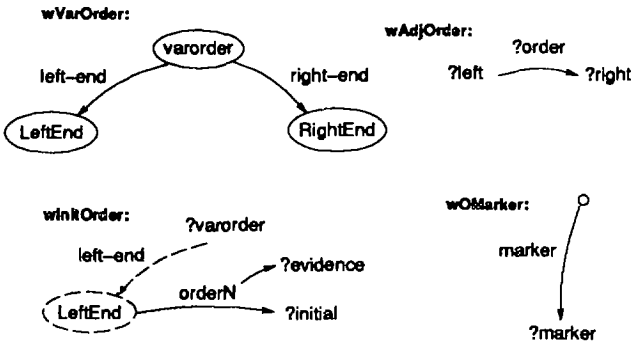
The overall order was obtained manually, but the process can be implemented using the topological sort algorithm [Cormen *et al.*, 1990]. This would be developed as an part of an application which is external to WEAVE and accessed through WEAVE's knowledge base manager.

Sharing Data Between Different Data Types

It is important not only to have a flexible method of defining data in the knowledge base, but also to be able to retrieve it in a manner which is most appropriate for the current task. We show one way of defining order in the knowledge base and two different ways for retrieving it. These three different views allow the knowledge base to be accessed for specific tasks to make explicit useful relationships and hide unwanted information.

One way of defining order is to use three types: VARORDER, ORDER, and OMARKER. Each element in VARORDER is composed of a collection of orders over OMarkers, where each OMarker is a set of MARKERS.

The elements in these types can be built using graph constructors which are roughly:



where the graph primitives denoted by dashed lines above already exist in the knowledge base, and the distinction between "left" and "right" is arbitrary.

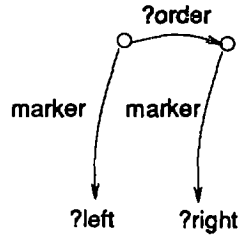
The ORDER data type can be defined by the knowledge base developer in SPIDER by:

```
defstype Order ()
  initorder(VarOrder,Evidence,OMarker) = wInitOrder
  adj(Order,OMarker,OMarker) = wAdj
```

and the other types can be defined similarly. This can be made more general by replacing the OMARKER data type with the user-defined SET(A) type constructor, then ORDER(A) is a type constructor which can create the ORDER(SET(MARKER)) data type.

These graph constructors are sufficient to build the order relationships, though different ones may be more useful depending upon the presentation of the order information in the external application and upon desired integrity constraints for the knowledge base. But, instead of looking at other graph constructors for defining order relationships, we will examine some of those useful for retrieving the order relationships.

Although it was useful to create a type such as OMARKER to refer to sets of markers when defining orders, it is more useful to refer to the individual markers when retrieving order information. This requires that the abstractions of the graph used for retrieval overlap multiple abstractions used for definition. For example, it is useful to see if there is an explicit order relationship between two markers. This abstraction can be defined as:



where *?left* and *?right* are bound in the query and all orders are returned in the result. This combines primitives which were used in both the ORDER and OMARKER types. The same graph abstraction can be used with, say, *?left* and *?order* bound to find the next marker in the order or with only *?left* bound to find any adjacent markers regardless of the source of the order information.

However, we want to isolate the end-user from dealing with the complexities of the WEB graphs, and we want to allow the application programs to access the data through SPIDER. To do this, the knowledge base developer must associate the graph constructors with data constructors to create new SPIDER types. There are at least two useful ways of stepping through an order relationship using SPIDER. One way is to use a data constructor `adjacent` which specifies that two chairs are adjacent. The data constructor `adjacent` would be associated with the graph constructor above. The second way is to have three data constructors `leftend`, `rightend`, and `interior` which access the markers using the following graph constructors:

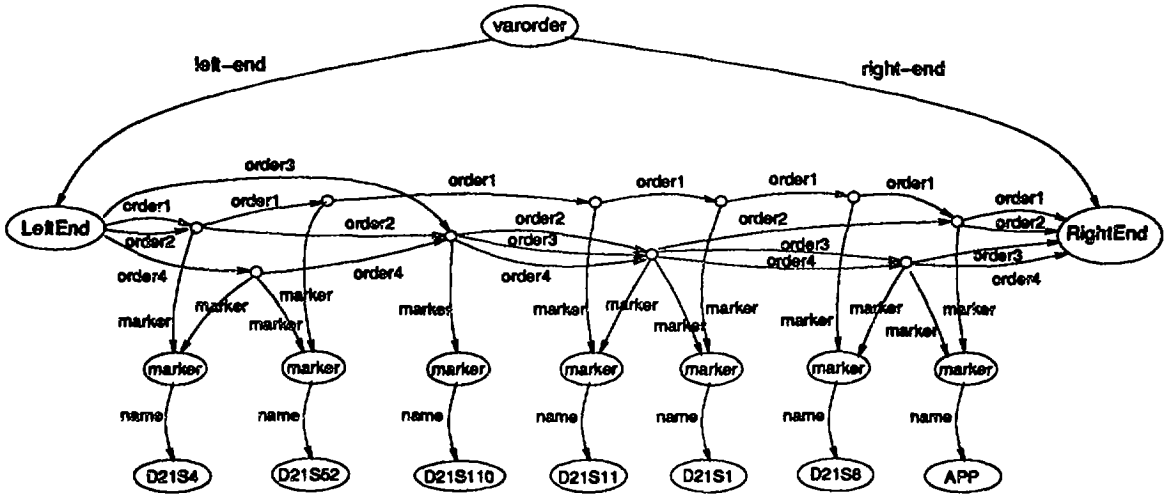
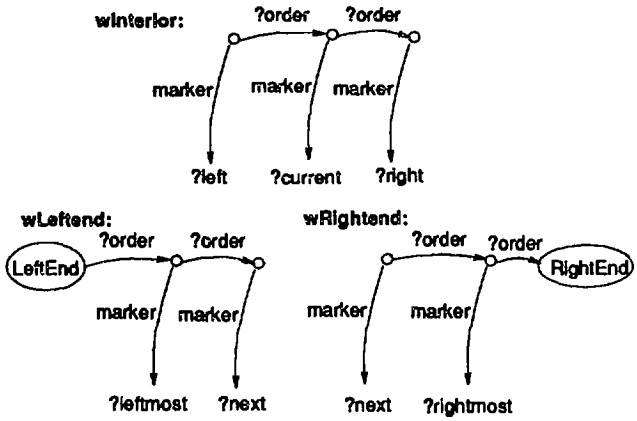


Figure 2: Order relationships from four different genome maps.

order num	map type	source	order
order1	Rad Hybrid	Cox 90	D21S4, D21S52, D21S11, D21S1, D21S8, APP
order2	Physical	Gardiner 92	D21S4, D21S110, $\begin{matrix}\text{D21S1}\\\text{D21S11}\end{matrix}$, APP
order3	Genetic	Warren 89	D21S110, $\begin{matrix}\text{D21S1}\\\text{D21S11}\end{matrix}$, $\begin{matrix}\text{APP}\\\text{D21S8}\end{matrix}$
order4	Genetic	Tanzi 92	$\begin{matrix}\text{D21S4}\\\text{D21S52}\end{matrix}$, D21S110, $\begin{matrix}\text{D21S1}\\\text{D21S11}\end{matrix}$, $\begin{matrix}\text{APP}\\\text{D21S8}\end{matrix}$
overall	—	—	D21S4, S52, S110, S11, S1, S8, APP

Table 1: Order relationships from four maps and their consensus order.



The markers can be accessed either using the `adjacent` data constructor (which is the simplest approach) or using the second approach (which provides more information to the application).

Access methods can be defined using SPIDER on these types for easier problem solving. A simple example is the function `left-of` which returns the seat to the left of the seat specified. This is defined by the developer as:

```
defsfun left-of MarkerOrder2(A)
        ()
  leftend(?order,?leftmost,?next) => :ERROR
  interior(?order,?left,?current,?right) => ?left
  rightend(?order,?rightmost,?next) => ?left
```

The advantage of WEAVE is that the type created for the definition of order and the different types created for access can be used without redundancy in the data. This occurs because the same graph primitives are used to define multiple data types.

Discussion

There are several advantages to designing a knowledge base to represent heterogeneous mapping information.

1. A knowledge base organizes the information in a clear, integrated framework which allows inferences to be made more easily.
2. There is now a process for designing knowledge bases which can guide development and make more efficient use of the map maker's time.

3. The formalisms we have described here have proven themselves expressive enough for a wide variety of tasks and appear sufficiently powerful to help solve the problem of integrating heterogeneous maps.
4. Because these formalisms are very flexible yet can be implemented efficiently, they promise to be an effective tool for mapping the human genome.

Acknowledgements

Thanks to Mike Boehnke, Clare Bates Congdon, Karen Mohlke, and Bill Rounds who commented on earlier versions of this paper. A portion of this work was supported by NSF grant ISI-9120851.

References

Aït-Kaci, Hassan and Podelski, Andreas 1991. Towards a meaning of LIFE. PRL Research Report 11, DEC Paris Research Lab.

Anderson, John R. 1985. *Cognitive Psychology and Its Implications.* W.H. Freeman and Co., 2nd edition.

Backhouse, Roland; Chisholm, Paul; and Malcolm, Grant 1988. Do-it-yourself type theory (parts 1 and 2). In *EATCS.*

Bic, Lubomir and Lee, Craig 1987. A data-driven model for a subset of logic programming. *ACM Transactions on Programming Languages and Systems* 9(4):618–645.

Brachman, R. J. and Schmolze, J. G. 1985. An overview of the KL-ONE knowledge representation system. *Cognitive Science* 171–216.

Brodie, Michael 1984. On data models. In Brodie, Michael; Mylopoulos, John; and Schmidt, Joachim, editors 1984, *On conceptual modelling: Perspectives from artificial intelligence, databases, and programming languages.* Springer-Verlag, New York. 19–47.

Carey, Micheal 1990. Extensible database management systems. *ACM SIGMOD Record* 19(4):54–60.

Constable, R.L.; Allen, S.F.; Bromley, H.M.; Cleaveland, W.R.; and others, 1986. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice Hall, Englewood Cliffs, NJ.

Cormen, Thomas H.; Leiserson, Charles E.; and Rivest, Ronald L. 1990. *Introduction to algorithms.* MIT Press.

Cox, DR; Burmeister, M; Price, ER; Kim, S; and Myers, RM 1990. Radiation hybrid mapping: a somatic cell genetic method for constructing high-resolution maps of mammalian chromosomes. *Science* 250:245–250.

Deliyanni, Amaryllis and Kowalski, R. 1979. Logic and semantic networks. *Communications of the ACM* 22(3):184–192.

Etherington, David; Borgida, Alex; Brachman, Ronald; and Kautz, Henry 1989. Vivid knowledge and tractable reasoning: Preliminary report. In *Proc. IJCAI-89.* IJCAI. 1146–1152.

Gardiner, K and Patterson, D 1992. The role of somatic cell hybrids in physical mapping. *Cytogenet Cell Genet* 59:82–85.

Graves, Mark 1993. *Theories and Tools for Designing Application-Specific Knowledge Base Data Models.* Ph.D. Dissertation, University of Michigan.

Kasper, R. T. and Rounds, W. C. 1986. A logical semantics for feature structures. In *Proceedings of the 24th Annual Conference of the Association for Computational Linguistics.* 235–242.

Martin-Löf, Per 1982. Constructive mathematics and computer programming. In *Sixth International Congress for Logic, Methodology, and Philosophy,* Amsterdam. North-Holland. 153–175.

Milner, Robin; Tofte, Mads; Harper, Robert; and Misbra, Prateek 1990. *The Definition of Standard ML.* MIT Press, Cambridge, MA.

Nebel, Bernhard and Smolka, Gert 1990. Representation and reasoning with attributive descriptions. In Bläsius, K. H.; Hedstück, U.; and Rollinger, C.-R., editors 1990, *Sorts and Types in Artificial Intelligence,* volume 418 of *LNAI.* Springer-Verlag. 112–139.

Ott, Jurg 1991. *Analysis of human genetic linkage.* Johns Hopkins University Press, Baltimore, revised edition.

Shieber, S. 1986. *An Introduction To Unification-Based Approaches To Grammar.* CSLI, Stanford, CA.

Sowa, J. F. 1984. *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley, Reading, MA.

Tanzi, Rudolph E; Haines, JL; Watkins, PC; Stewart, GD; Wallace, MR; Hallewell, R; Wong, C; Wexler, NS; Conneally, PM; and Gusella, JF 1988. Genetic linkage map of human chromosome 21. *Genomics* 3:129–136.

Tanzi, Rudolph E; Watkins, PC; Stewart, GD; Wexler, NS; Gusella, JF; and Haines, JL 1992. A genetic linkage map of human chromosome 21: Analysis of recombination as a function of sex and age. *American Journal Human Genetics* 551–558.

Warren, Andrew C; Slaugenhaupt, SA; Lewis, JG; Chakravarti, A; and Antonarakis, SE 1989. A genetic linkage map of 17 markers on human chromosome 21. *Genomics* 4:579–591.

Zdonik, Stanley B. and Maier, David, editors 1990. *Readings in object-oriented database systems.* Morgan Kaufmann, San Mateo, CA.