AN INFORMATION MODEL FOR GENOME MAP REPRESENTATION AND ASSEMBLY[1]

A. J. Lee*, E. A. Rundensteiner*, S. Thomas**, and S. Lafortune*

(*) Dept. of Electrical Eng. and Computer Science
University of Michigan, Ann Arbor
Ann Arbor, MI 48109-2122
e-mail: rundenst@eecs.umich.edu
telephone: (313) 936-2971
fax: (313) 763-1503

(**) Information Technology and Networking
University of Michigan Medical Center
Human Genome Center
2570C MSRB II
Ann Arbor, MI 48109-0674
e-mail: spencer@eecs.umich.edu
telephone: (313) 764-8065
fax: (313) 764-4133

May 1993

Abstract

In this paper, we focus on some of the scientific data management problems faced by the Human Genome Project. In particular, we describe the design of an information model for the physical contig map assembly task. First, we present an object-oriented data schema that captures scientific genomic data and their relationships required for this task, including the raw experimental data and the derived data gotten through scientific analysis. Our genome object representation efficiently supports the maintenance of unordered, partially ordered, and completely ordered sets of data based on an overlap refinement hierarchy of interval relationships. We describe operators we have developed to automate analysis steps currently performed manually by the scientists. Examples are operators for inverting local frames of orientation, for combining information in different frames into another more informative one, and for inferring additional overlap information using transitivity rules. In conclusion, we provide a walk-through example that demonstrates how our approach can be used to effectively support some of the experiments currently performed by the scientists at the Michigan Human Genome Center.

Keywords: Ordering abstraction hierarchy, interval inferencing, relative orientation, scientific databases, object-oriented databases.

---

[1]The authors are grateful to acknowledge partial financial support from the NIH under grant NIH #P30HG00209-03.

1 INTRODUCTION

1.1 The Human Genome Application Domain

The human genome project is an international 15-year effort to map all the genes in the human genome and to use that information for diagnosis and treatment of genetic diseases. There are roughly 100,000 genes in each human cell that make up the human genome. Each gene encodes information about how and when to form the proteins that govern the daily functioning of the human body, and ranges in size from 2,000 to 2 million base pairs long. The physical mapping process [2] first separates the chromosomes from the nucleus, then breaks them up into smaller pieces, which are inserted into the genome of lower organisms, e.g., bacteria, where they can be replicated rapidly and in large amounts. The resulting DNA fragments (clones) are then used for future experiments. The purpose of the experiments is to identify features of the DNA fragments and determine the locations of these features on a chromosome by ordering the fragments to their respective locations on the chromosomes. Then, genes can be isolated from clones, sequenced to get the genetic code, and finally the code can be used to help determine the function of the protein product. Knowing this may lead to diagnosis and treatment for genetic diseases such as cystic fibrosis.

Due to this urgency of understanding genetic information, and its likelihood to contribute in a fundamental way to our understanding of nature, there is increased excitement nationwide in investigating these and related issues. A result of this increased effort is an explosion with respect to the amount of information generated on a daily basis. The development of more and more sophisticated experimental techniques to introduce and analyze more data furthermore compound this problem. According to a recent report of the National Research Council: "The mapping and sequencing effort will generate more data than any other single project in the history of biology. For example, just to record the 3 billion nucleotides of the human genome will require nearly 1 million pages of printed text [7]." This increasing volume of data provides diverse challenges for computer scientists and system engineers.

In short, the scope and complexity of the Human Genome Project is such that previously acceptable manual methods are no longer suitable. It thus is vital that we develop intelligent software systems that manage genetic data by providing suitable representations for objects applicable to genetics. Database design is one important component of this effort, with a goal of developing databases that give geneticists friendly, correct, and efficient access to the human genome database in a user-friendly fashion. This project focuses on building an information model for genome physical map representation and assembly. The system promises to be a useful tool for the geneticists of the Human Genome Center at the University of Michigan Medical Center.

1.2 Our Approach

In this paper, we present an information model that captures the genomic objects and interrelationships for supporting the physical map assembly task, which range from raw experimental data over analytically derived data up to metadata describing, for instance, the data sources.

Our model is developed using object-oriented database (OODB) technology [3] since the latter provides powerful modeling constructs for capturing such complex genomic information naturally and efficiently.

We are interested in creating a *contig map*, assembling overlapping DNA fragments into a contiguous stretch. Therefore, we must represent known and derive additional information about the order and position of sets of DNA fragments. The concept of ordering and orientation is fundamental to our system. Our genome object schema efficiently supports the maintenance of unordered, partially ordered, and completely ordered sets of data based on an overlap refinement hierarchy. The relative ordering information we get from experiments may not be orientable with respect to the global ordering viewed at the chromosome level. Our model thus supports the concept of a local orientation frame which represents the relative orientation of an ordering with respect to the global view. Additionally, we describe sets of operators we have developed to automate some analysis steps. Examples are operators for combining information in different frames into a single frame, and for inferring additional overlap information using transitivity rules.

Limitations of current systems are a lack of support for concepts of distance and ordering, inability to accommodate accuracy and inconsistency of experimental results, and poor support for handling derived data. The overall goals of our work can thus be summarized as follows:

- Support the efficient representation of both precisely and partially known ordering data;
- Provide for the identification of inconsistent information in ordering relationships and the potential of resolution of such situations;
- Develop an object model that integrates both experimental and derived data;
- Allow incremental development of the data base as new experimental data is added; and
- Support the effective coupling of a database with analytical tools, such that the analysis of the data and the derivation of the physical map can be directly supported within the context of our intelligent database system.

1.3 Organization of Paper

The rest of the paper is structured as follows. We discuss related work in Section 2. In Section 3, we introduce necessary terminology of molecular genetics, a translation into computer related terms, and the core of an object model for a genome database. Sections 4 and 5 describe our ordering model and associated set of operators, respectively. Section 6 gives an overview of the system architecture, Section 7 goes through a comprehensive example, and Section 8 discusses conclusions and future work.

2 RELATED WORK

Traditional databases do not capture well the complexity of scientific data; therefore we propose to design a genome object model using OODB technology [8]. This technology, which has emerged recently, provides powerful modeling constructs capable of capturing complex information. In this paper, we identify the modeling requirements of genome data as needed for the map assembly task, and then explore how these can be satisfied by OODB modeling. While relational databases are used by most genome projects, Goodman [8] also advocates utilizing OODB technology in place of more traditional database technology for the informatics support of genome mapping projects. The focus of Goodman's work [8] is on genetic rather than physical map construction. In [10], Honda et al. describe an object model for genome data that covers several levels of resolution, including genome maps, DNA sequences, and gene sequences.

In genomics, we can view both DNA fragments and probes as intervals, which have neither precise location information nor known distal extent. Thus, we find the work in the temporal reasoning domain relevant to our project. Temporal reasoning deals with events and relationships between events, which are intervals with variable or unknown starting and ending times. Allen [1] lists all possible relationships between temporal intervals assuming complete information about their durations, and presents a transitivity table for these relationships.

In the temporal domain there is a simple global orientation. Once we know event A happened before event B, the ordering is global. This is different from the genome domain, where we are more likely to be able to infer only relative ordering information. Experiments are performed against DNA fragments rather than the whole chromosome, so we can actually make statements that are true only in some local frame of reference. Secondly, we cannot generally determine the exact position of intervals and their endpoints. Thus, we may determine that two intervals overlap, but not whether their left endpoints are identical. We have developed an abstraction hierarchy of ordering relationships for the genomic domain which accounts for these characteristics. The abstraction hierarchy naturally describes the amount of knowledge we know about the endpoint relationships of the two intervals.

Letovsky and Berlyn [11] use local ordering windows to represent such relative orderings. Their work deals only with points rather than intervals. We combine the 'interval' concept [1] and the 'window' idea [11] to solve the DNA fragment ordering problem.

Guidi and Roderick [9] survey issues required of intelligent systems to support research efforts in genomics. In particular, they focus on inference of order, including ambiguity and uncertainty, as one of the more important issues that need to be addressed. In the remainder of this paper, we outline our approach for addressing this problem.

3 THE OBJECT MODEL FOR MAP ASSEMBLY

In this section, we will introduce the genetic terminology used throughout the remainder of this paper, and relate it to concepts familiar to computer scientists. This can best be done by briefly introducing the core of the object model that we have developed to model the genetic data of the map assembly domain (Figure 1).

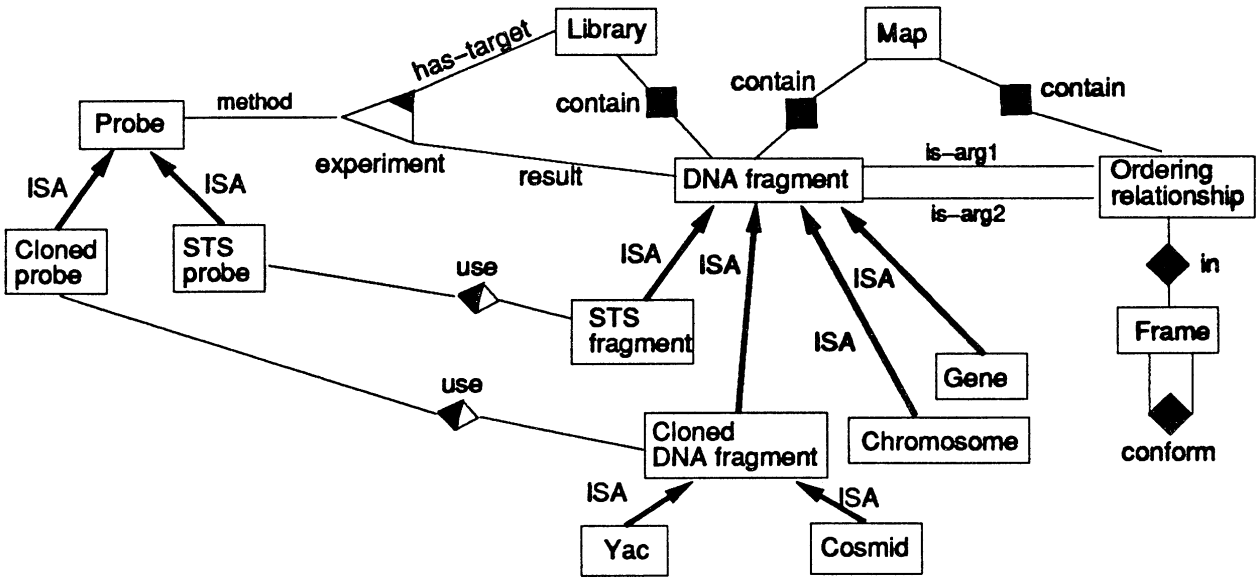


Figure 1: The Map Assembly Object Model.

3.1 Introducing Genomic Terminology

The total genetic material present in each cell is called the genome. The smallest genetic unit, a nucleotide, is also called a base and takes on one of the 4 values: A, C, G, T. The human genome has about 3 billion bases. If we view a nucleotide as a letter, then the human genome can be viewed as a 3 giga byte book written in 4 letters. DNA is the polymeric molecule that stores genetic information, and it is double-stranded. The bases of the two strands match according to the following rules : A matches with T, and C matches with G. A–T and C–G are called base pairs, or in short, bases. A DNA fragment is a contiguous piece of DNA, and thus can be viewed as a string of letters.

There are different types of DNA fragments, e.g., chromosomes, genes, STS fragments, and clones. There are 23 pairs of chromosomes in a human cell, and each chromosome contains many genes. The average size of a human chromosome is 150 Mbp, and it can be viewed as a chapter in a book. Genes are functional units that are responsible for producing proteins used by the human body. A gene may range from a few 100 to 1,000 base pairs long, and can be viewed as a sentence. Some genes have been identified that cause certain diseases when they are faulty. A STS (sequence tagged site) is a short DNA fragment, which is a site in the genome that can be uniquely identified by a short sequence of bases. It can be viewed as a word, except it usually has no functional meaning. A clone is a colony of organisms with identical genetic makeup, or the DNA fragment of interest from such a colony. A cloned DNA fragment can be sub-classified into classes such as YAC (yeast artificial chromosome) and Cosmid. A Cosmid, about 40 to 80 kilo base pairs long, is cloned in bacteria, and can be viewed as a paragraph. A Yac, about 100 to 1000 kilo base pairs long, is cloned in yeast, and can be viewed as a longer paragraph or section in a chapter. In addition, there are other words and sentences, and cosmids and YACs may overlap.

A STS or a cloned DNA fragment can be used as a probe against a target library in an experiment. A library is an unordered set of DNA fragments, usually clones. A probe is a labeled DNA sequence used to detect the presence of a complementary sequence, e.g., by

molecular hybridization. Hybridization is an experiment that compares two DNA fragments to see if they have some sequence in common. A physical map corresponds to a (partially) ordered set of DNA fragments forming a particular region of a chromosome. We can also view the genomic map assembly task as a string matching task. The main problem is that the geneticists must build the genomic map without being able to read each letter (base pair) in the DNA fragment, because it is too expensive to do so.

3.2 The Map Assembly Object Model

We have identified the following classes for our map assembly object model, each of which corresponds to independent entities in the domain: Library, DNA fragment, Probe, Experiment, Ordering relationship, Frame, Locus, and Map. A library is a collection of DNA fragments. An experiment represents the relationship among three classes: a probe, a target library, and a resulting set of DNA fragments. The probe and the target library are inputs to an experiment, and the resulting set is its output. One type of experiments is shown in Figure 2. STSs with some known ordering information are used to probe a YAC library. Ordering of the YACs and additional ordering of the STSs may be derived.

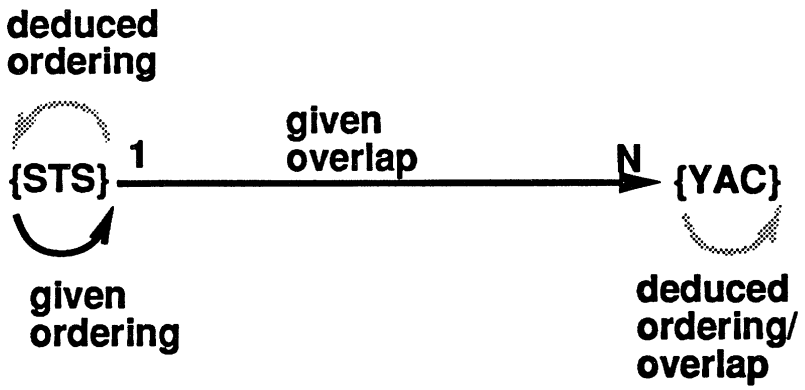


Figure 2: A Map Assembly Experiment.

The binary ordering relationship between two DNA fragments is represented as an object. It may come from experiments, from external sources, or may be derived by the system. In this paper, we classify ordering relationships for genomic intervals using an ordering refinement hierarchy. A detailed discussion of our approach toward map assembly using this overlap abstraction concept will be given in Section 4.

The overlap and ordering relationships of DNA fragments are specified within some local orientation for the following reasons: first, a chromosome is broken into pieces and we thus can not tell which side of a DNA fragment is left or right with respect to the chromosome, and second, the experiments are done on a collection of DNA fragments and we thus will only determine relationship relative to this respective collection. Since this local orientation may not conform with the global orientation, we introduce the concept of a local orientation frame which allows us to represent knowledge about that a set of relationships exhibit the same orientation, without having to specify whether this orientation corresponds to the original orientation of the complete chromosome. An ordering relationship with respect to a local orientation frame represents a relative ordering in the global view. A detailed discussion of this can be found in Section 4.

4 THE ORDERING REPRESENTATION MODEL

4.1 The Abstraction Hierarchy of Interval Relationships

We want to determine the order and position of overlapping DNA fragments spanning the entire chromosomal region. Prior work in ordering of intervals has focused on temporal ordering [1, 6]. While some concepts from temporal ordering can apply to DNA fragment ordering, there are several significant differences:

- First, in genomic applications, the determination of the equality of endpoint relationships between DNA fragments is generally too costly. Hence, Allen's temporal relationships that refer to equality of endpoints are not applicable to our problem, these are the relationships *meets*, *starts*, *ends* and their inverses. However, we are interested in determining as much as possible about the interval relationships, i.e., relationships between endpoints will be either *before*, *after*, or *unknown*.
- Second, information in genomic applications will initially be imprecise but will gain precision as more experiments are performed. For example, if DNA fragments A and B are both hit by probe STS1, then we know A and B overlap. However, we do not know which of A and B starts first or which ends first. Allen expresses a precise temporal relationship between two events as a simple relationship, whereas a partially known temporal relationship has to be expressed as a disjunction of precise relationships. In short, the less we know about a relationship between two intervals, the more complex Allen's representation becomes. In the following, we propose a model that represents imprecise relationships as concisely as precise relationships.
- Third, we rarely experimentally determine ordering relationships between two DNA fragments directly. Rather we determine overlaps that can be used to infer such relationships. Map assembly applications typically determine ordering relationships between intervals (e.g., YACs) based on their overlap with point-like intervals (e.g., STSs). We therefore want to integrate into our model, in a natural way, the interactions between point and interval relationships.

We have developed a refinement hierarchy of interval relationships, shown in Figure 3, that addresses the issues raised above [2]. It is based on our analysis of the type of experiments conducted with DNA fragments. For example, if we know that two intervals A and B share one common probe P1, then we know that A and B are not disjoint, denoted by *not-disjoint-1(A,B)* in our model. If we know that in addition there is one probe P2 that hits interval A but not interval B, then we know that A and B are not disjoint and that A cannot be contained in B, denoted by *not-disjoint<(A,B)*. Full details on the meaning of the relationships can be found in Appendix A. The key features of this abstraction hierarchy can be summarized as follows:

---

[2] We use the following convention for expressing relationships between two intervals A and B: For each relationship rel(A,B), we have either (1) *start(A) < start(B)*, (2) *end(A) < end(B)*, or (3) *start(A) < end(B)*. (1) has the highest priority, (2) the next highest, and (3) has the lowest priority. We denote the start and end points of an interval A by start(A) and end(A), respectively.

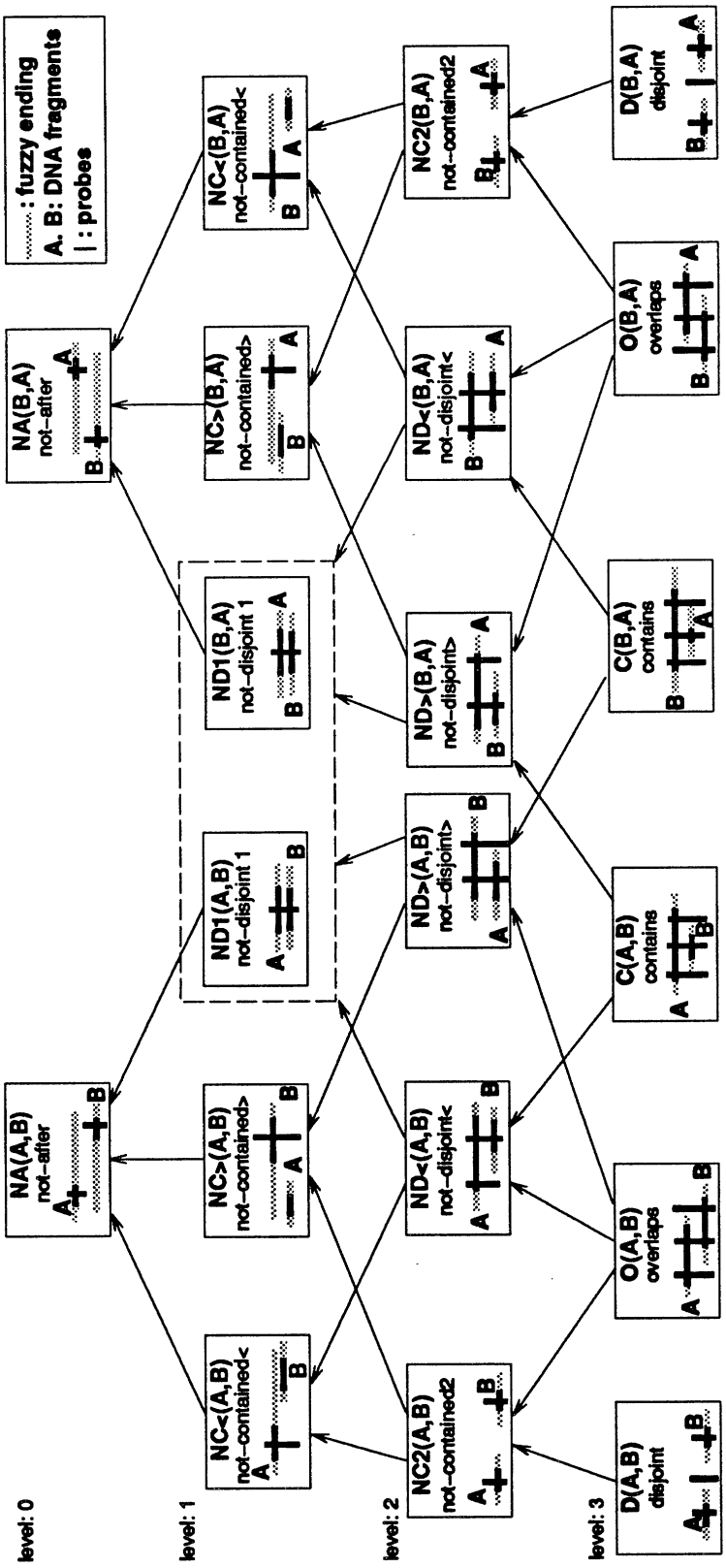


Figure 3: The Abstraction Hierarchy of Interval Relationships.

- The representation supports both *precise* as well as *partial* information about interval relationships. The hierarchy naturally describes the amount of knowledge we know about the endpoint relationships of the two intervals. In general, interval relationships at the i-th level of the abstraction hierarchy are characterized by the fact that exactly (i+1) of their relative endpoint relationships are known. For example, exactly two endpoint relationships are known for the interval relationships at level 1. The interval relationships at level 3 are characterized by four endpoint relationships. This corresponds to precise information, since a relationship between two intervals is completely described by the four relationships between their endpoints.

- The representation of interval relationships is *uniform* such that both precisely and partially known relationships between two intervals can be represented with the same descriptive complexity, i.e., a single relationship.

- The representation is *complete.* Namely, the composition of any combination of these interval relationships using transitive closure operators will again result in one of the relationships contained in the abstraction hierarchy.

- The hierarchy naturally captures a notion of the minimum amount of effort involved in establishing these interval relationships. Namely, one probe is required to identify the relationships on level 1 of the hierarchy. We need at least two probes for the relationships on level 2 of the hierarchy, and three probes for the relationships on level 3. Our representation model thus captures the process of incrementally increasing our knowledge about the interval relationships using experimental data.

4.2 The Local Orientation Frame Concept

Unlike in the temporal domain where we deal with a strict time line, the ordering information about DNA fragments may often be deduced without knowing their global orientation with respect to the chromosome. In our model, the overlap and ordering relationships of DNA fragments are specified in local orientation frames for the following reasons: first, a chromosome is broken into pieces and we thus cannot tell which end of a DNA fragment is left or right with respect to the chromosome, and second, the experiments are done on a collection of DNA fragments and we thus will only determine relationships relative to this respective collection. Thus, we introduce the concept of a local orientation frame, which allows us to represent that a set of relationships exhibit the same orientation, without having to specify whether this orientation corresponds to the original orientation of the complete chromosome. As shown in Figure 4, knowing that a DNA fragment B is located between fragments A and C does not automatically tell us what is the true global orientation [11], which could be either "A before B before C" or "C before B before A".

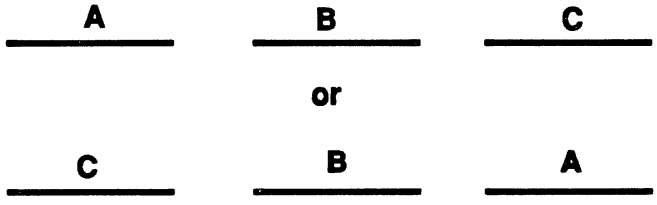
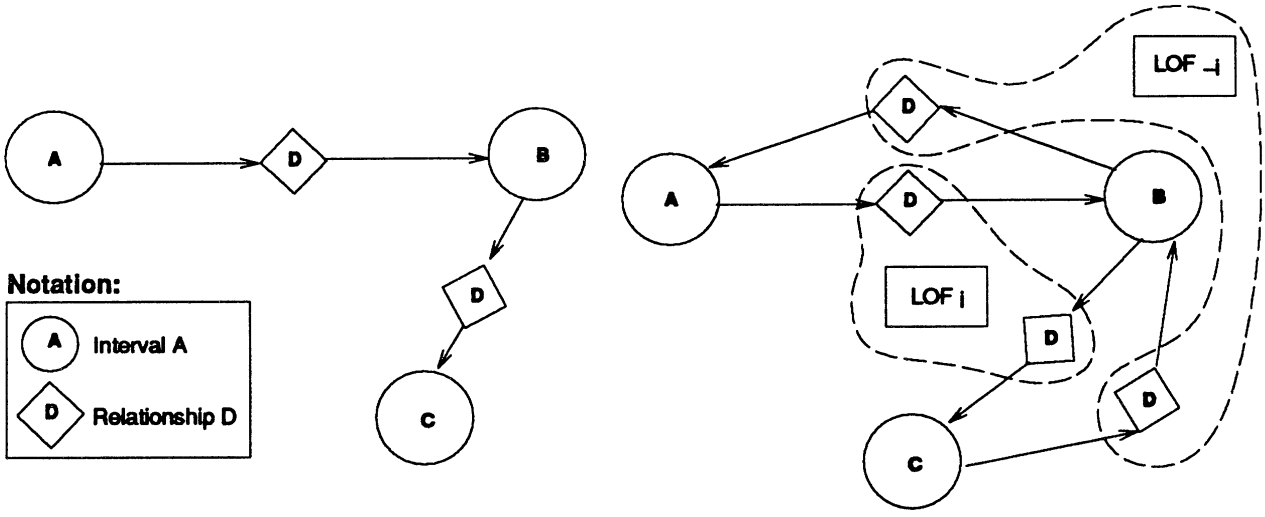


Figure 4: The Local Orientation Frame Concept.

We express an ordering relationship between two DNA fragments as a pairwise constraint between fragments (as typically done in constraint propagation systems). This allows us to map our genomic model to a simple graph representation. We construct a graph by mapping each DNA fragment to a node of the graph and each ordering relationship between fragments to an arc connecting the two respective nodes. For instance, the relationship (A before B before C) will be represented by the graph in Figure 5.a.

If we want to represent ((A before B before C) or (C before B before A)), then we need to be able to capture the associations among the four binary relationships in the graph structure. For this purpose, we introduce a grouping construct that combines each set of interval relationships which are known to have the same orientation. This grouping construct corresponds to the *local orientation frame* (LOF) introduced above (Figure 5.b). An ordering relationship is now said to belong to a *local orientation frame.* For instance, we would describe the situation in Figure 5.b by [*"A disjoint B" in LOF$_i$*] and [*"B disjoint C" in LOF$_i$*]. The *local orientation frame* is similar in spirit to the orientation windows introduced in [11] for orderings between points.



(a) Graph Representation of Overlap Constraints. (b) Graph Representation of LOFs.

Figure 5: Graph Representation of a Local Orientation Frame.

For a given *local orientation frame*, LOF$_i$, we may not know which of its two possible orientations is correct. Therefore, we construct the mirror image of a LOF$_i$, denoted by LOF$_{-i}$. Including the mirror images into the system will allow us to apply inferencing rules without having to explicitly consider the inverse cases, thus simplifying our inference operators. The mirror image of LOF$_i$ is constructed by first building the mirror image of each individual interval relationship and then by combining them into a new frame, LOF$_{-i}$. For instance, for point relationships, if [*"A before B" in LOF$_i$*] then [*"B before A" in LOF$_{-i}$*]. For interval relationships, we need to determine the inverse of each of the relationships. In The inverse of each relationship can be determined (1) by inverting the arguments of the relationship (e.g., from (A,B) to (B,A)) and (2) by replacing the relationship by its inverse (e.g., replace ND< by ND>). For a complete listing of mirror image construction see Appendix A.

4.3 Supporting Evidence

We further extend our model by characterizing the interval relationships in terms of the evidence of *experimental data* used to deduce them. For instance, the *not-disjoint-1*(A,B) relationship can be established between the interval A and B as soon as we have identified one probe P that is located both on A and on B. We thus annotate the relationship *not-disjoint-1*(A,B) by the probe P. For the identification of the *overlap*(A,B) relationship, for instance, we may need three properly placed probes. This direct annotation of the interval relationships with their supporting evidence serves several purposes: (i) to enable our system to perform inferencing more efficiently based on the evidence and (ii) to capture a measure of certainty in the derived interval relationship. The more evidence we have found supporting a particular relationship, the more likely the relationship is to hold when dealing with the presence and resolution of inconsistent information. Eventually, the maintenance of supporting evidence will also support the retraction of derived information, if one of the sources is proven incorrect, as well as the generation of explanations about how we get to the current state of our genomic knowledge.

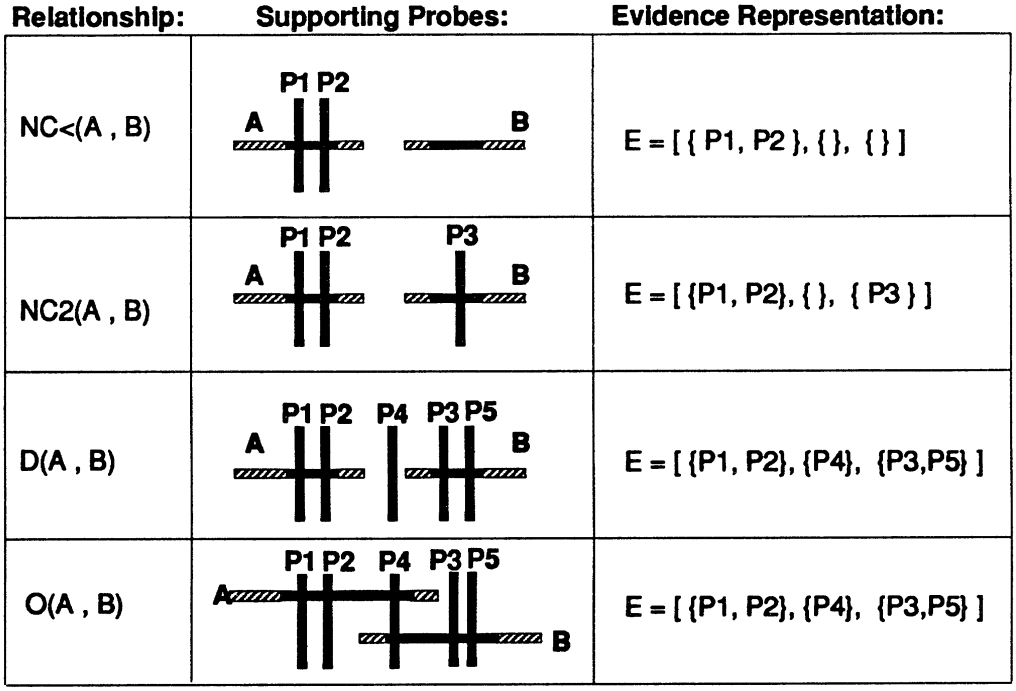


Figure 6: Examples of Constructing the Evidence for an Interval Relationship.

We adopt the following representation scheme for capturing the supporting evidence:

evidence(rel_i(A,B)) = [S1, S2, S3]

with S1, S2, S3 possibly empty sets of probes that characterize the rel_i relationship. The set S1 contains all probes that support the known information of rel_i with respect to the left side of the pair A and B, i.e., overlap on A and not on B. The set S2 contains all probes that support the known information of rel_i with respect to the middle of the pair A and B, i.e., overlap on both or disjointness between them. The set S3 contains all probes that support the known information of rel_i with respect to the right side of the pair A and B, i.e., overlap on

B and not on A. Figure 6 depicts several examples that demonstrate how these three sets are constructed for some of the interval relationships.

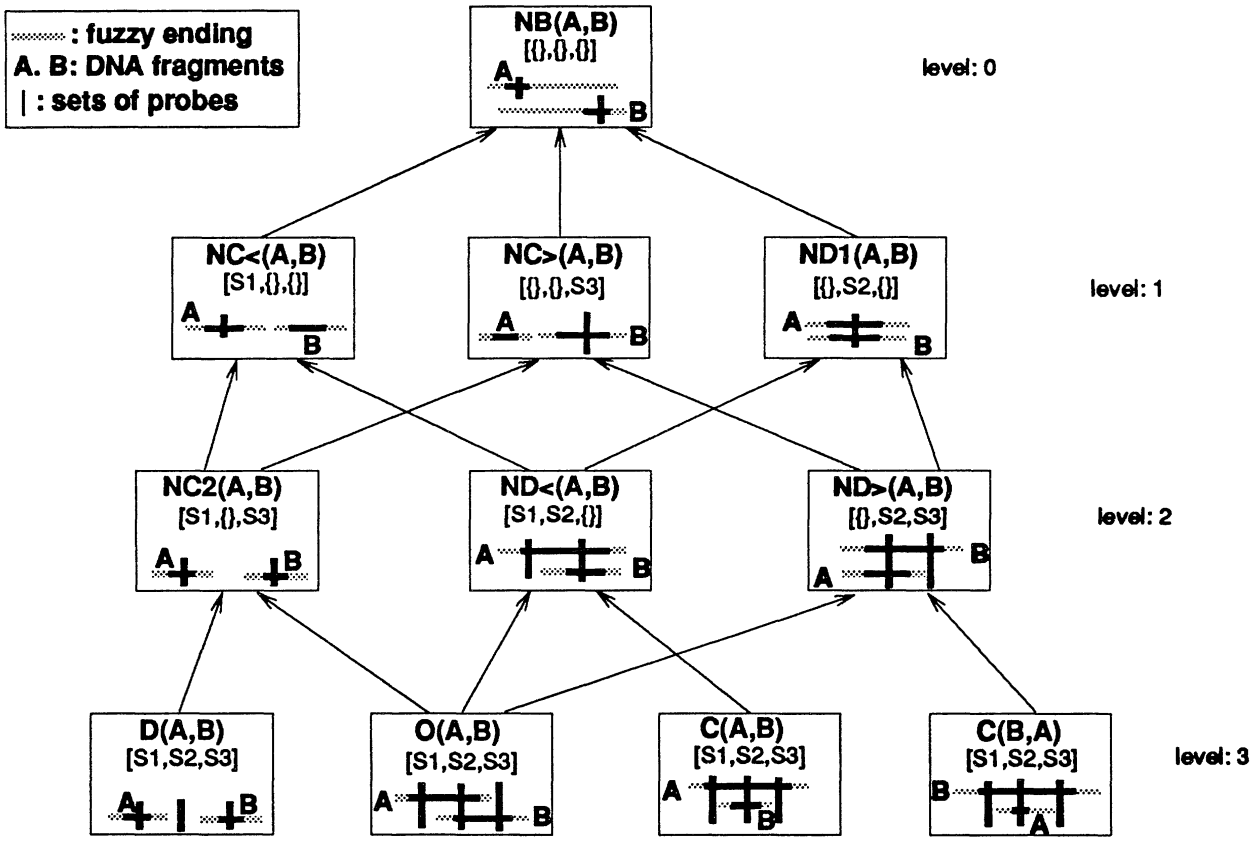


Figure 7: Extending the Interval Abstraction Hierarchy with Supporting Evidence (depicting only one half of the symmetric abstraction hierarchy from Figure 3).

Figure 7 depicts the complete abstraction hierarchy extended with the evidence representation. The evidence-based inference rules provide us a mechanism to do interval-to-point and also point-to-interval inferencing, as shown in Section 5.3. The explicit capture of evidence also allows us to easily combine LOFs without any search for supporting probes.

5 INTERVAL ORDERING OPERATORS

In this section, we describe the different sets of operators that we have developed for our genomic representation model.

5.1 The Composition of Local Orientation Frames

As discussed in the previous section, we group interval relationships that are known to have the same orientation into local orientation frames. This separation into different LOFs allows us to explicitly represent uncertainty about how one set of ordering constraints is oriented relative

to another one. For example, the ordering relationships [*"A before B before C" in* LOF_i] and [*"C before D before E" in* LOF_j] with $i \neq j$, would mean that we do not know whether LOF_i and LOF_j represent the same orientation or not (Figure 8.a). If it turns out that LOF_i and LOF_j have the same orientation then we could reduce this to one total ordering (Figure 8.b). If they have the opposite orientation, then numerous total orderings are possible (Figure 8.c).

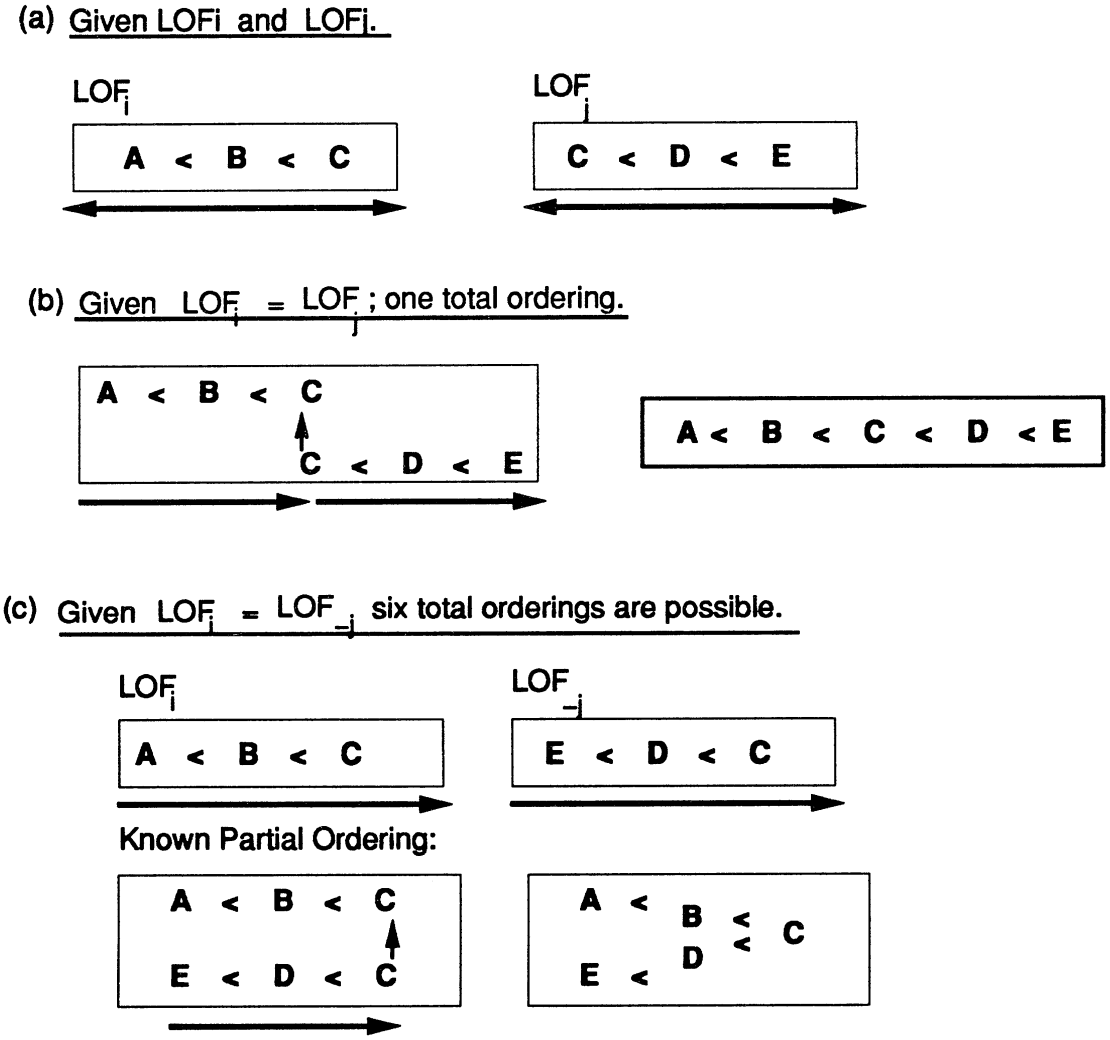


Figure 8: An Example of the LOF Representation.

It is our goal to derive as completely as possible the ordering relationships between fragments. Hence, we want to combine frames into more informative sets, whenever possible. Ideally, all relationships would be combined into *one* frame with fixed orientation with respect to the chromosome, the *global orientation frame*. As can be seen in Figure 8.a, LOF_i and LOF_j cannot be combined even though they share the common element C. Generally speaking, two LOFs can be merged whenever it can be determined that they both represent the same orientation. At a minimum, this requires the presence of the same pair of objects (A,B) in the two LOFs with relationship constraints between them expressing orientation rather than just overlap information. For ordering relationships between points [11], we simply would need to identify one pair of points (A,B) that belongs to both LOFs and that has an ordering relationship in each LOF, i.e., (A *before* B) or (B *before* A). See Figure 9.a for an example demonstrating

this situation. For intervals, on the other hand, the situation is somewhat more complicated since the types of interval relationships experienced in the two LOFs become significant. This is best explained using an example.

Example 1 *To merge two LOFs containing ordering relationships between intervals, we need to identify one pair of intervals that belongs to both LOFs, say the pair (A,B) in Example 1 in Figure 9.b. Next, we need to determine whether the relationships of the pair (A,B) in* LOF_i *and in* LOF_j *together fix the same orientation. Since not-disjoint-<(A,B) and not-disjoint->(A,B) have a common subclass in the abstraction hierarchy (Figure 3), i.e., they refined to overlap(A,B), the two LOFs can now be combined into one LOF, denoted by* $LOF_{i,j}$. *By inspection, it can be seen that the overlap(A,B) relationship must hold in the combined* $LOF_{i,j}$.

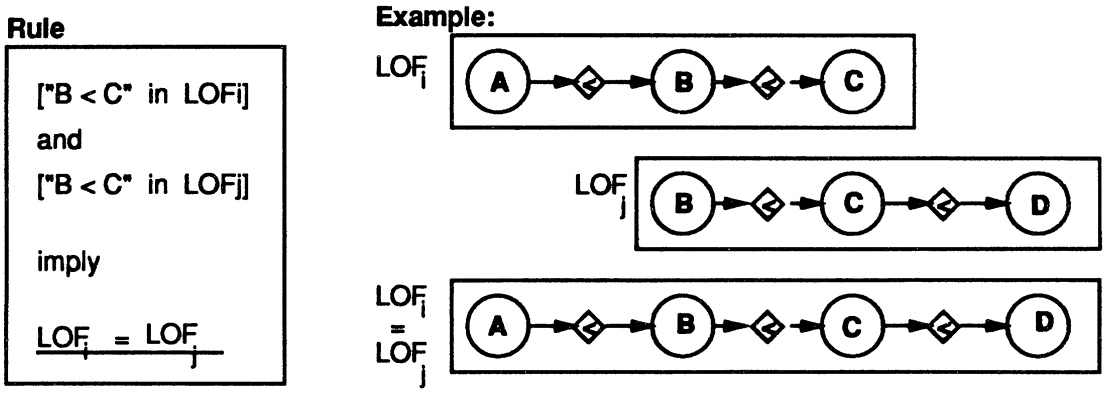
As demonstrated by the example, there are essentially two types of mechanisms involved in constructing the LOF composition rules:

- First, we need to determine whether two given relationships together fix the same orientation, i.e., whether these two relationships have at least one common subclass and all of their common subclasses in the abstraction hierarchy fix the orientation.
- Second, we need to determine how to combine any pair of interval relationships with the same arguments into a single relationship for the new frame.

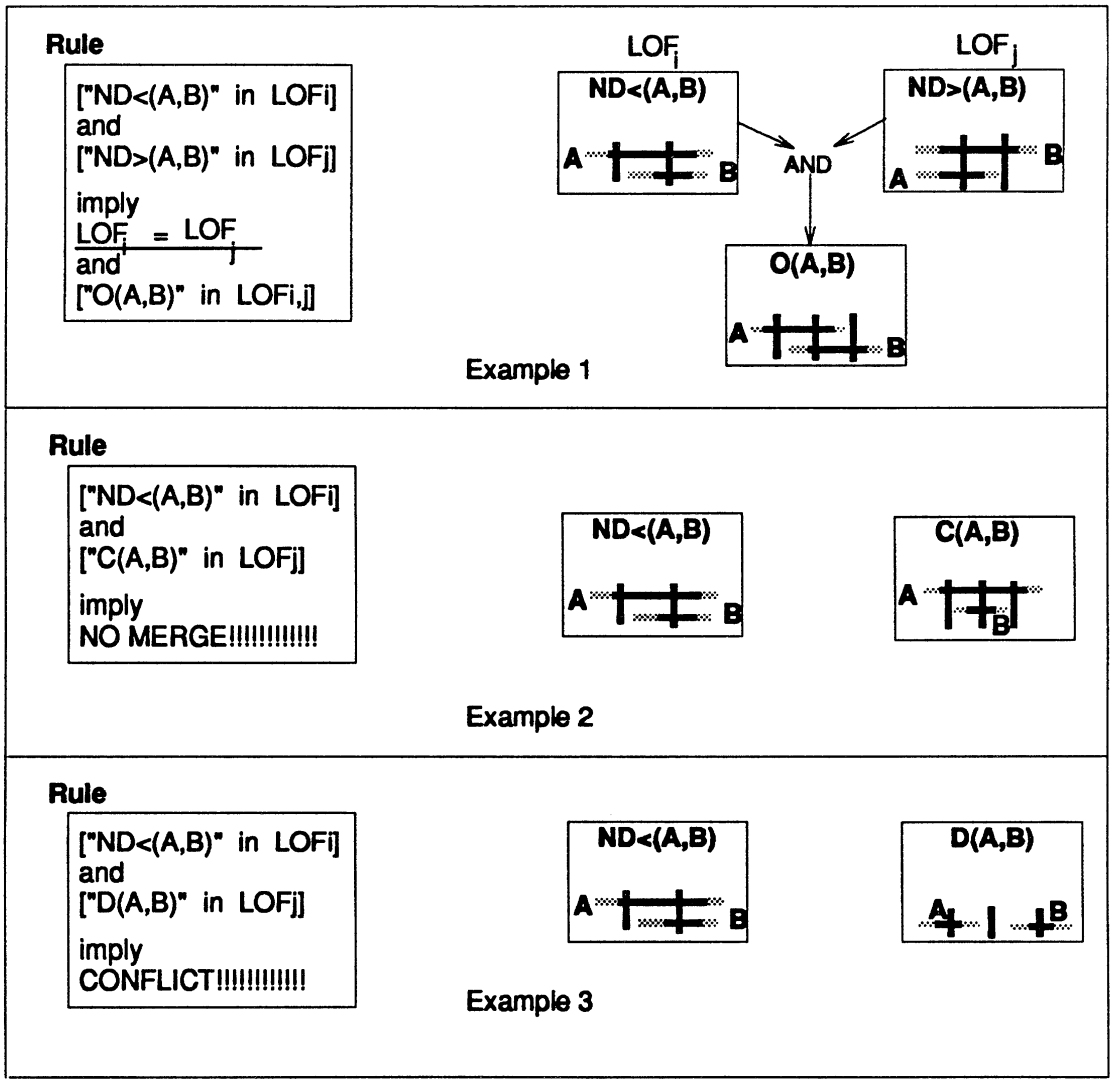
The first issue can be resolved by studying whether the two input relationships together fix the orientation between their two arguments. There are three relationships that do not provide any orientation information, namely, *contains*, *not-disjoint-1*, and *not-after*. For example, *contains(A,B)* does not by itself fix the orientation, since the relationship could be placed into two frames LOF_i and LOF_{-i} without causing contradictions. Examples of relationships that do fix the orientation are *overlaps(A,B)*, *disjoint(A,B)* and *not-contained-2(A,B)*. In Example 1 in Figure 9.b, the two LOFs can be combined because both relationships express a common orientation. In Example 2 in Figure 9.b, the two LOFs cannot be combined because it could lead to a non-orientable relationship *contains(A,B)*.

The second issue, the determination of the resulting interval relationship, is computed by a graph traversal algorithm of the abstraction overlap hierarchy given in Figure 3. Namely, given two relationships rel1(A,B) and rel2(A,B) in the now combined LOF, we set rel3(A,B) := rel1(A,B) $\wedge$ rel2(A,B) equal to the highest common node below rel1 and rel2 in the hierarchy. As shown in Figure 3, *overlap(A,B)* is the only, hence the highest, node below *not-disjoint-<(A,B)* and *not-disjoint->(A,B)*. Hence the LOF_i and LOF_j in Example 1 in Figure 9.b can be merged and the relationship between A and B after composition should be *overlap(A,B)*.

If, on the other hand, no such common lower node exists in the abstraction hierarchy, then the two relationships express conflicting information. For instance, in Example 3 in Figure 9.b, there is no common node below the two relationship nodes *not-disjoint-<(A,B)* and *disjoint(A,B)* in the abstraction hierarchy. The two relationships clearly are in conflict. While at present our model simply identifies these conflicts in the data, in the future this conflict identification can serve as foundation for conflict resolution strategies.



(a) For Ordering Relationships between Points.



(b) For Ordering Relationships between Intervals.

Figure 9: Examples of Combining Local Orientation Frames.

The complete table of rules for combining two LOFs (including whether or not merging can take place and what the resulting relationship would be in the new LOF) can be found in Appendix B. In general, we can utilize the LOF composition table for two purposes: one, to increase our knowledge about interval relationships by merging LOFs and two, to discover inconsistencies between the experimental data stored in different LOFs.

5.2 Transitivity Rules

Transitivity rules are used to infer additional ordering information about intervals in a single LOF. The transitivity rule for point-based ordering says that if (A before B) and (B before C) in the same LOF then (A before C) in that LOF. We now develop similar rules for the composition of our interval relationships. Namely, we develop rules of the following type: *"(A rel1 B) and (B rel2 C) implies (A rel3 C)"*. One example is shown in Figure 10; the complete table of transitivity rules is given in Appendix C.

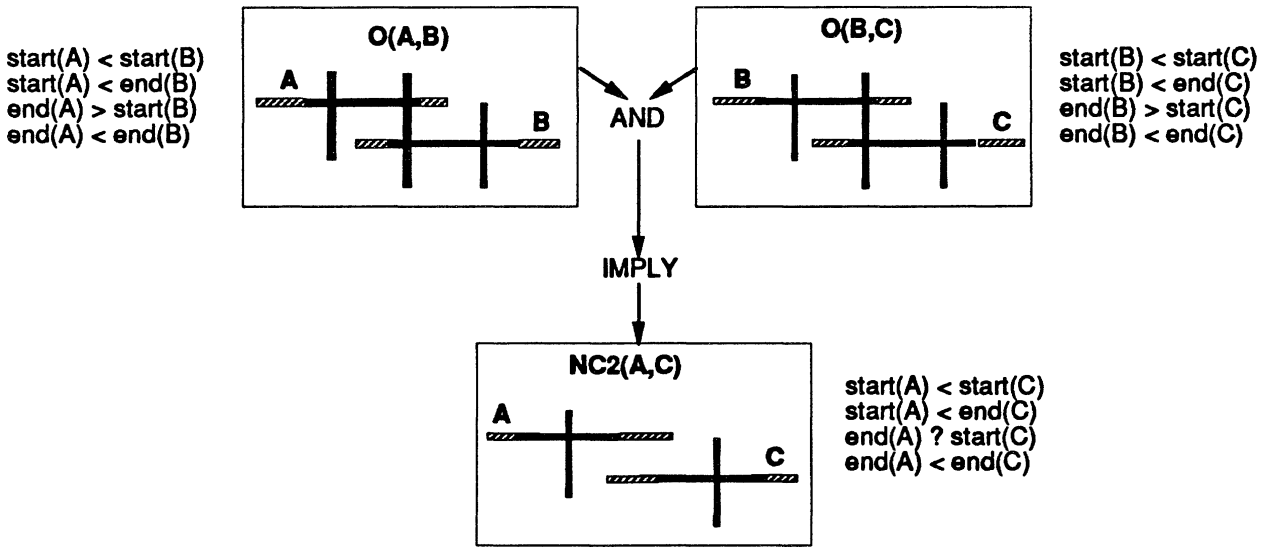


Figure 10: An Example of a Transitivity Rule for Interval Relationships.

The transitivity rules in Appendix C are derived based on the known endpoint relationships between the respective intervals. This is also illustrated in Figure 10. The transitivity table demonstrates the completeness and uniformity of our representation. Namely, the composition of any combination of our interval relationships results in one of the interval relationship representations contained in the abstraction hierarchy, i.e., no need for disjunctions arises. Furthermore, the imprecise information is expressed as efficiently as the precise information, namely, with one relationship each. Consequently, the result of an inference will always be equal to one relationship, assuming consistent data.

5.3 Rules for Evidence-Based Inferencing

As discussed in Section 4.3, our representation of interval relationships includes the supporting evidence of *experimental data*. In this section, we describe how the evidence is propagated

through the inference rules. First, we present a rule for propagating evidence through the LOF composition rules, i.e., for combining evidence of two relationships about the same pair of intervals (A,B).

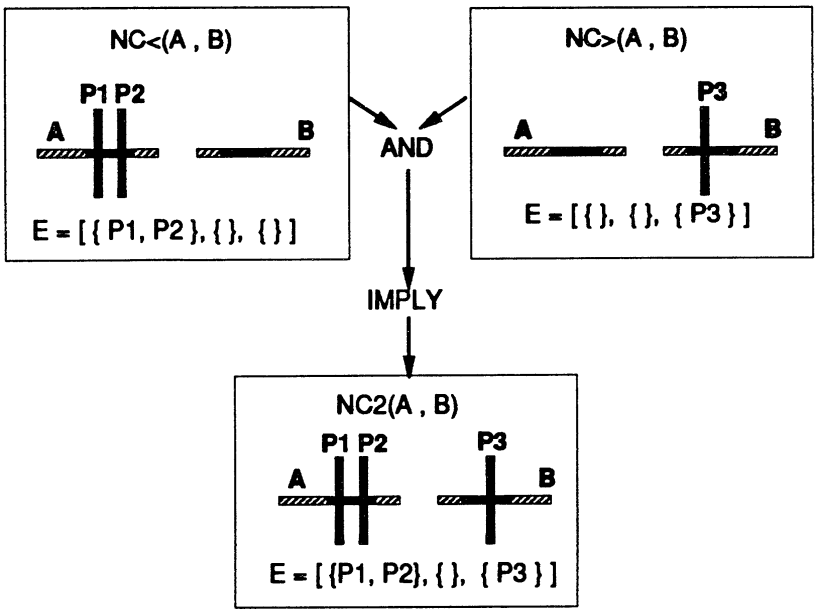


Figure 11: An Example of Combining Evidence for a LOF Composition Rule.

Rule 1 *Given rel1(A,B) with evidence(rel1(A,B)) = [S_{11}, S_{12}, S_{13}] and rel2(A,B) with evidence(rel2(A,B)) = [S_{21}, S_{22}, S_{23}], then the supporting evidence for rel3(A,B) := (rel1(A,B) • rel2(A,B)) is computed by evidence(rel3(A,B)) = [$S_{11} \cup S_{21}$, $S_{12} \cup S_{22}$, $S_{13} \cup S_{23}$], with "•" the LOF composition rules given in Appendix B.*

Below, we discuss an example of the application of Rule 1 for combining evidence associated with the input arguments for a LOF composition rule.

Example 2 *This example is based on Figure 11. If we apply the LOF composition rule to derive the relationship not-contained-2(A,B) := not-contained-<(A,B) • not-contained->(A,B), then the evidence of the resulting relationship is computed by building a union of the respective evidence sets. Assuming evidence(not-contained-<(A,B)) = [S_{11}, S_{12}, S_{13}] and evidence(not-contained->(A,B)) = [S_{21}, S_{22}, S_{23}], then evidence(not-contained-2(A,B) = [$S_{11} \cup S_{21}$, $S_{12} \cup S_{22}$, $S_{13} \cup S_{23}$] = [{ P1,P2 } ∪ {}, { } ∪ {}, { } ∪ {P3}] = [{ P1,P2 }, {}, {P3}].*

We also define rules for propagating the supporting evidence through the transitivity rules. Unfortunately, there appears to be no generic rule that covers all cases, as for the LOF composition rules. Rather, a different formula must be developed for each transitivity rule, i.e., for each entry in the transitivity table in Appendix C is given in Appendix D.

Example 3 *By Appendix D, the evidence-composition formula for the example depicted in Figure 12, i.e., the transitivity rule disjoint(A,C) := disjoint(A,B) o [3] not-contained-2(B,C) is*

[3] o denotes the operator of the transitivity rule given in Appendix C.

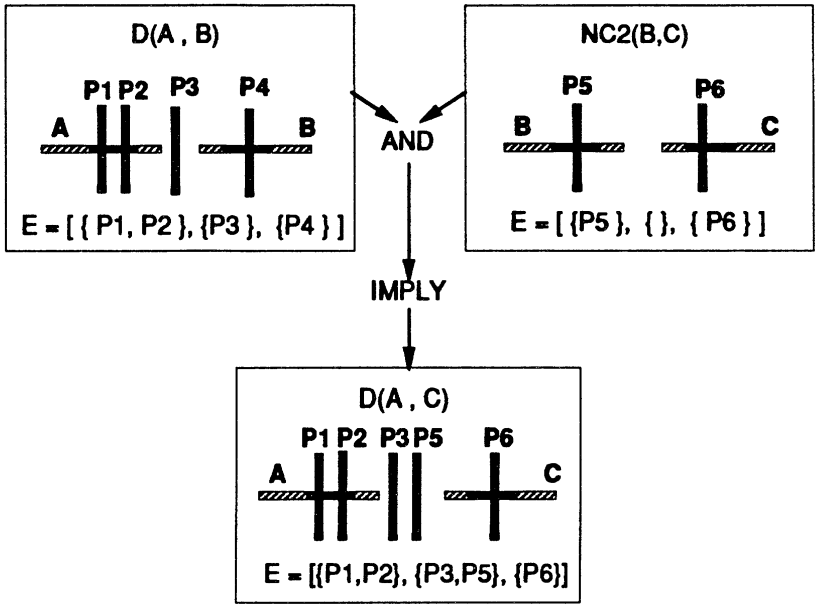


Figure 12: An Example of Propagating Evidence through Transitivity Rules.

evidence(disjoint(A,C)) = [S_{11}, $S_{12} \cup S_{21}$, S_{23}]. In Figure 12, the resulting evidence would thus be evidence(disjoint(A,C)) := [{ P1,P2 }, {P3}, { P4 }] $\oplus$ [4] *[{P5}, { }, {P6}] = [{ P1,P2 } ∪ {}, { P3 } ∪ {P5}, { } ∪ {P6}] = [{ P1,P2 }, {P3, P5}, {P6}].*

Finally, we can establish the following rule for deriving point to point and point to interval relationships from interval relationships which have been annotated by evidence.

Rule 2 *Given rel1(A,B) in LOF_i with evidence(rel1(A,B)) = [S1, S2, S3], then we can deduce the addition of the following relationships into LOF_i*[5]*: ∀ s_j ∈ S1, ∀ s_k ∈ S2, ∀ s_l ∈ S3, (1) add the following point to point relationships: disjoint(s_j,s_k), disjoint(s_k,s_l), and disjoint(s_j,s_l); (2) add the following point to interval relationships: contains(A,s_j), disjoint(A,s_l), contains(B,s_l), and disjoint(s_j,B); (3) if (rel1 = disjoint) then add disjoint(A,s_k) and disjoint(s_k,B); else add contains(A,s_k) and contains(B,s_k).*

Rule 2 is straightforward, since it is directly derived from the semantics of the evidence sets as explained in Section 4.2. This rule then forms the foundation for merging distinct LOFs, based on relationships between points.

5.4 Contig Map Construction

Our system will derive the maximal information about the ordering relationships stored in the database, given the operators described in the previous section. While this forms the

[4] $\oplus$ denotes the operator of the evidence-combination rule given in Appendix D.

[5] We use the interval relationship *disjoint*(s_i,s_j) as defined in Figure 3 to denote the "before" relationship between points, i.e., $s_i < s_j$. Similarly, we use the *disjoint*(interval,point) relationship to indicate that the point is not located on but rather after the interval, while *contains*(interval,point) indicates that the point is completely positioned within the interval.

foundation of discovering further relationships, it would be too confusing to present the fully elaborated information to the user. For example, for a set of n completely ordered disjoint intervals, the resulting transitive closure would contain $n \times (n-1)/2$ disjoint relationships. It is sufficient, however, to give the $n-1$ relationships between adjacent intervals. Thus, the redundant relationships should be pruned.

One way to simplify information is *minimal contig* construction. The goals of minimal contig construction are: 1) to cover the *largest possible contiguous regions* of the chromosome and 2) to use the *minimal number* of intervals needed to construct the map.

We have made the following assumptions for contig map construction:

1. Maps are built by using known connectivity. That is, if we don't know whether two DNA fragments overlap or not, we make the pessimistic assumption that they do not. Therefore, contig maps are based on the *overlaps*, *not-disjoint-<*, and *not-disjoint->* relationships only. *Not-disjoint-1* and *contains* relationships cannot extend the map, and thus they are not used.
2. We assume all DNA fragments in a library will have similar lengths. Hence, for *not-disjoint->(A,B)*, we assume that the DNA fragment A is before the DNA fragment B in the contig, and vice versa, for *not-disjoint-<(A,B)*.

We have designed a simple algorithm for minimal contig construction that performs the following tasks in a local orientation frame (lof) to return contig(s) with more than one DNA fragment in each contig:

```
For each LOF in the database do:
1. Build a subgraph G (a directed acyclic graph) for the LOF consisting only
   of ND>, ND<, and O relationships and their associated intervals.
   The orientation of the edge corresponding to each relationship R(A,B)
   is from the left argument A to the right argument B.
2. While nodes in G are not marked:
   a. From a left-most interval (node), use breadth-first search to mark
      each connected node with its shortest distance from the starting node.
   b. Extract a minimal path between the starting node and a right-most
      connected node; this is a contig.
3. Order the contigs generated by step 2, if possible, using the disjoint
   relationships between their members.
```

An example of a minimal contig constructed by our algorithm is depicted in Figure 14 in Section 7.

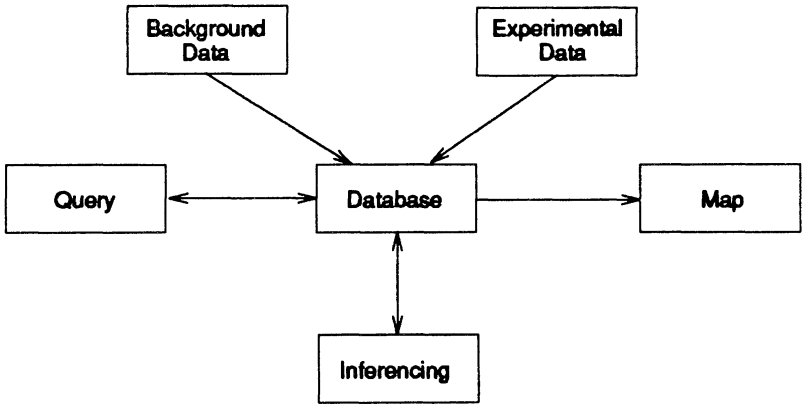


Figure 13: The System Architecture

6 THE GENOME DATABASE PROTOTYPE

We are currently building a database prototype of the system described in this paper using the GemStone database system, the OODB product marketed by Servio (with an initial prototype based on a simple point-based overlap representation already being completed)

The block diagram of our system is given in Figure 13. Background data and experimental data are inputs to our system. Background data may contain the information about libraries and the clones in the libraries, experimental device characteristics, documentation, and data dictionary. Experimental data is the information about which probes are used against which target libraries and what are the resulting set of DNA fragments that are intersected with the probes. The inferencing module computes the transitive closure and combines LOFs that have the same orientation. The derived information is stored back into the database, and it may trigger more inferencing rules to be fired. The query module allows the users to ask questions about the data, and the map module generates preliminary physical maps.

7 A WALK-THROUGH EXAMPLE

In this section, we will briefly describe one example of applying our approach to a physical map assembly problem. The goal is to demonstrate how our system can be utilized to get from the initial experimental data to the desired contig map.

For this example we assume the following experimental data:

```
1.probe 1 hits YAC y1
2.probe 2 hits YACs y1, y2, y3
3.probe 3 hits YACs y3, y4, y5, y6
4.probe 4 hits YACs y4, y7
5.probe 5 hits YACs y7, y8
```

Translation of experimental data into the internal representation:

The experimental data is translated using the following rules: for each YAC y that is hit by a probe p create a relationship *contains(y,p)*; for each yac y in the target library that is not hit by the probe p create a relationship *disjoint(y,p)*. Each containment constraint is placed into the global orientation frame, while each disjoint constraint is placed into a new LOF. For example, the input from the third experiment will be translated into the following representation: [contains(y3,p3), contains(y4,p3), contains(y5,p3), contains(y6,p3) in LOF_{global}], [disjoint(p3,y1) in LOF_i], [disjoint(p3,y2) in LOF_j], [disjoint(p3,y7) in LOF_k], [disjoint(p3,y8) in LOF_l].

Ordering Inferencing:

After we apply the inferencing rules to combine LOFs and compute the transitive closure, our system will merge all relationships into one LOF:

```
[ND<(y1,y2)  O(y1,y3)    NC2(y1,y4)  NC2(y1,y5)  NC2(y1,y6)  D(y1,y7)
D(y1,y8)     ND>(y2,y3)  NC2(y2,y4)  NC2(y2,y5)  NC2(y2,y6)  D(y2,y7)
D(y2,y8)     O(y3,y4)    ND<(y3,y5)  ND<(y3,y6)  NC2(y3,y7)  D(y3,y8)
ND>(y5,y4)   ND>(y6,y4)  O(y4,y7)    NC2(y4,y8)  ND1(y5,y6)  NC2(y5,y7)
D(y5,y8)     NC2(y6,y7)  D(y6,y8)    ND<(y7,y8) in LOFi].
```

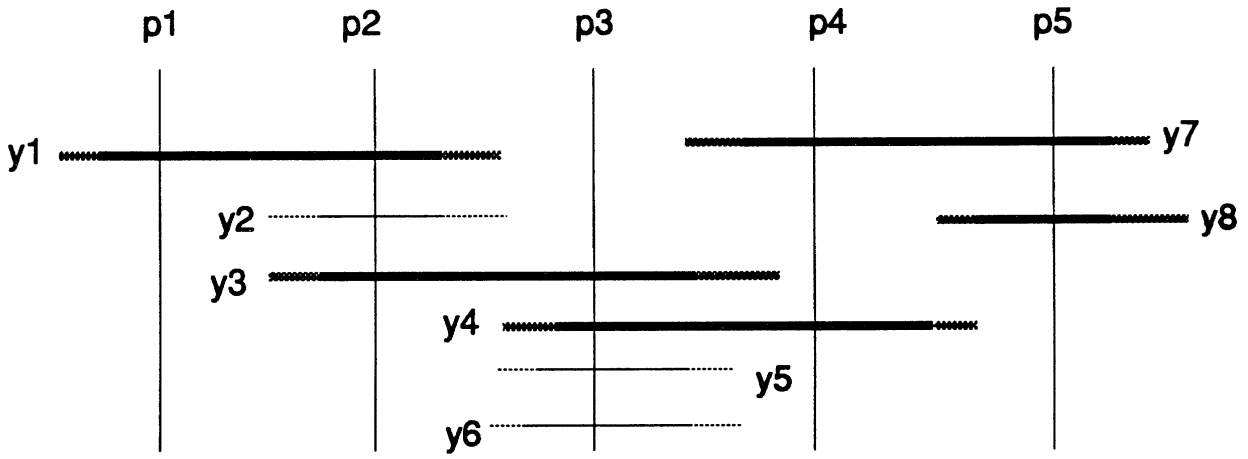


Figure 14: The Contig y1-y3-y4-y7-y8.

Map Generation:

To build the contig(s), we consider only relationships with known connectivity. That is we build the contig(s) using the following overlapping relationships: ND<(y1,y2), O(y1,y3), ND>(y2,y3), O(y3,y4), ND<(y3,y5), ND<(y3,y6),

ND>(y5,y4), ND>(y6,y4), O(y4,y7), and ND<(y7,y8). Using the algorithm described in Section 5.4, we construct the contig "y1-y3-y4-y7-y8" depicted in Figure 14. The figure shows the final contig (the boldface lines) as well as the set of overlapping relationships generated by our system. The fragment y2 is not used in the contig, because it is covered by y1 ∪ y3. For similar reasons y5 and y6 are not used. Based on assumption 2, y8 is the right most DNA fragment in the contig.

8 CONCLUSIONS AND FUTURE WORK

8.1 Conclusions

In this paper, we have described the ordering information model, which we designed as part of our effort in developing an intelligent OODB system for the Human Genome Center at the University of Michigan Medical Center. The system will allow the scientists to store and analyze experimental data. Use of our system in conjunction with other tools will speed up many of the data intensive tasks of the Human Genome Project currently performed manually.

A key piece of our model is the abstraction hierarchy of the overlapping relationships of DNA fragments. It supports the natural representation both of the imprecise and precise overlapping relationships typical of the genome domain. Because the abstraction hierarchy is uniform and complete, it represents overlapping knowledge in a more compact format than previously possible in other systems.

Because a chromosome is chopped into pieces to perform experiments, the ordering relations we get from the experiments do not necessarily represent the global ordering. Hence, we have added a local orientation frame to each ordering relation to represent the concept of the relative ordering. Combining interval ordering relations with orientations allows us to correctly represent and manipulate genomic knowledge.

Based on our preliminary experience with this project, we found the OODB paradigm to be suitable for implementing a genome database for several reasons: its flexibility to easily change our system as the genome domain is evolving, its powerful constructs capable of modeling complex genomic data and their interrelationships, its computational completeness allowing us to directly operate on the persistent data structures and thus avoiding the impedance mismatch problem, and lastly its support for genome objects to play different roles in the database and to be modeled at varying levels of details [10].

8.2 Future Work

In order to support the full range of capabilities required by the Human Genome Project, there are many areas that need to be further investigated. For instance, we need to extend our current information model, which supports partial knowledge about ordering, to also handle inaccurate and conflicting experimental data. For a truely effective usage of our system by scientists we also need to replace our simple user interface with a more appropriate I/O paradigm, i.e., research into visualization methods is needed. The development of an easy to use, yet powerful, query interface to process arbitrary queries is another challenging open problem. Lastly, as the rule set is becoming larger, a general framework of integrating rules into the OODB system needs to be developed.

References

[1] Allen, J. F., "Maintaining Knowledge About Temporal Intervals", CACM, November 1983, Volume 26, pp 832-843.

[2] Carrano, A. V. et al., "Constructing Chromosome- And Region-Specific Cosmid Maps Of The Human Genome", Genome, 31, PP. 1059-1065, 1989.

[3] Cattell, R. G. G.,"Object data management : object-oriented and extended relational database systems", Addison-Wesley, 1991.

[4] Cormen, T. H., Leiserson, F. E., and Rivest, R. L., "Introduction To Algorithms", pp 469-476, 1991.

[5] Evans, G. A., "Combinatoric Strategies For Genome Mapping", BioEssays, Vol. 13, No. 1, January 1991.

[6] Freksa, C., "Temporal Reasoning Based On Semi-Intervals", Artificial Intelligence 54, pp 199-227, 1992.

[7] French, J. C. Jones, A. K., and Pfaltz, J. L., "Scientific Database Management", University of Virginia, Technical Report 90-21, Dept. of Computer Science, Aug. 1990.

[8] Frenkel, K. A., "The Human Genome Project and Informatics", CACM, Nov. 91, Vol. 34, No. 11, pp 41-51.

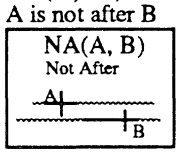
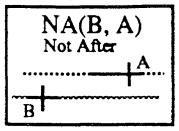
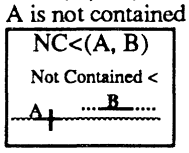
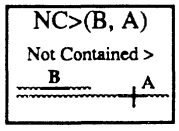
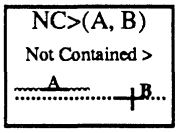
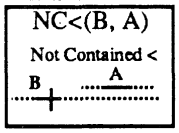
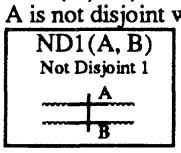
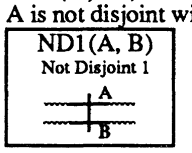
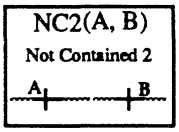
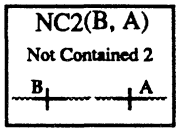
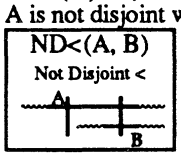
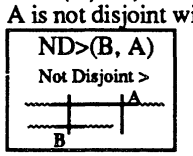
[9] Guidi, J. N. and T. H. Roderick, "Inference of Order in Genetic Systems," The First Internat. Conf. on Intelligent Systems for Molecular Biology, July 1993.

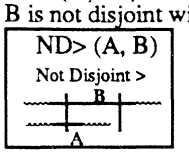
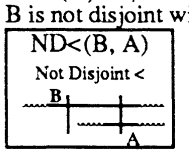
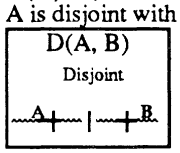
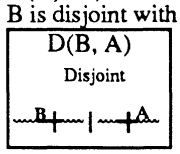
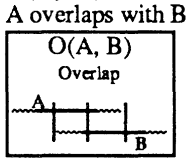
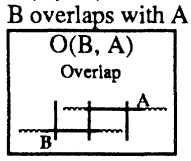
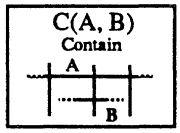
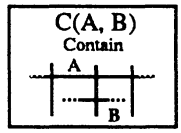
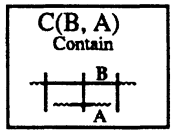
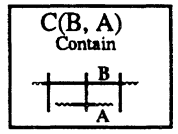
[10] Honda, S., Parrot, N. W., Smith, R., and Lawrence, C., "An Object Model for Genome Information at All Levels of Resolution," Proc. of 26th Annual Hawai Internat. Conf. on System Sciences, Vol. I, pp. 564 - 573, 1993.

[11] Letovsky, S. and Berlyn, M. B., "CPROP: A Rule-Based Program For Constructing Genetic Maps", Genomics 12, pp 435-446, 1992.

[12] Pecherer, R., "Contig Graph Tool: A Graphical Interface For Contig Physical Map Assembly", Hawai Int. Conf. on System Sciences, Vol. I, pp. 544-553, 1993.

A. Table of Frame Def. and Inversion

LOFi	LOF-i
NA(A, B) A is not after B	**NA(B, A)** B is not after A
 s(A) ? s(B) s(A) < e(B) e(A) ? s(B) e(A) ? e(B)	
(1 endpoint relationship)	
NC<(A, B) A is not contained in B	**NC>(B, A)** A is not contained in B
 s(A) < s(B) s(A) < e(B) e(A) ? s(B) e(A) ? e(B)	
(2 endpoint relationships, although 1 relationship is redundant.)	
NC>(A, B) B is not contained in A	**NC<(B, A)** B is not contained in A
 s(A) ? s(B) s(A) < e(B) e(A) ? s(B) e(A) < e(B)	
(2 endpoint relationships, although 1 relationship is redundant.)	
ND1(A, B) A is not disjoint with B	**ND1(A, B)** A is not disjoint with B
 s(A) ? s(B) s(A) < e(B) e(A) > s(B) e(A) ? e(B)	
(2 endpoint relationships, and both are essential.)	
NC2(A, B) A is not contained in B and B is not contained in A	**NC2(B, A)** A is not contained in B and B is not contained in A
 s(A) < s(B) s(A) < e(B) e(A) ? s(B) e(A) < e(B)	
(3 endpoint relationships, and one is redundant.)	
ND<(A, B) A is not disjoint with B	**ND>(B, A)** A is not disjoint with B
 s(A) < s(B) s(A) < e(B) e(A) > s(B) e(A) ? e(B)	
(3 endpoint relationships, and one is redundant.)	

LOFi	LOF-i
ND>(A, B) B is not disjoint with A	**ND<(B, A)** B is not disjoint with A
 s(A) ? s(B) s(A) < e(B) e(A) > s(B) e(A) < e(B)	
(3 endpoint relationships, and one is redundant.)	
D(A, B) A is disjoint with B	**D(B, A)** B is disjoint with A
 s(A) < s(B) s(A) < e(B) e(A) < s(B) e(A) < e(B)	
(4 endpoint relationships, and 3 are redundant.)	
O(A, B) A overlaps with B	**O(B, A)** B overlaps with A
 s(A) < s(B) s(A) < e(B) e(A) > s(B) e(A) < e(B)	
(4 endpoint relationships, and 1 is redundant.)	
C(A, B) A contains B	**C(A, B)** A contains B
 s(A) < s(B) s(A) < e(B) e(A) > s(B) e(A) > e(B)	
(4 endpoint relationships, and 2 are redundant.)	
C(B, A) B contains A	**C(B, A)** B contains A
 s(A) > s(B) s(A) < e(B) e(A) > s(B) e(A) < e(B)	
(4 endpoint relationships, and 2 are redundant.)	

able of Frame Composition Inference Rules

LOFj / i	NC<(A,B)	NC>(A,B)	ND1(A,B)	NC2(A,B)	ND<(A,B)	ND>(A,B)	D(A,B)	O(A,B)	C(A,B)	C(B,A)
:(A,B)	No	LOFi=LOFj NC2	No NC< -> ND<	LOFi=LOFj NC2	No NC< -> ND<	LOFi=LOFj O	LOFi=LOFj D	LOFi=LOFj O	No NC< -> C	Conflict
·(A,B)	LOFi=LOFj NC2	No	No NC> -> ND>	LOFi=LOFj NC2	LOFi=LOFj O	No NC> -> ND>	LOFi=LOFj D	LOFi=LOFj O	Conflict	No NC< -> C
(A,B)	No NC< -> ND<	No NC> -> ND>	No	No NC2 -> O	No	No	Conflict	No	No ND1 -> C	No ND1 -> C
:(A,B)	LOFi=LOFj NC2	LOFi=LOFj NC2	No NC2 -> O	LOFi=LOFj	LOFi=LOFj O	LOFi=LOFj O	LOFi=LOFj D	LOFi=LOFj O	Conflict	Conflict
:(A,B)	No NC< -> ND<	LOFi=LOFj O	No	LOFi=LOFj O	No	LOFi=LOFj O	Conflict	LOFi=LOFj O	No ND< -> C	Conflict
·(A,B)	LOFi=LOFj O	No NC> -> ND>	No	LOFi=LOFj O	LOFi=LOFj O	No	Conflict	LOFi=LOFj O	Conflict	No ND> -> C
B)	LOFi=LOFj D	LOFi=LOFj D	Conflict	LOFi=LOFj D	Conflict	Conflict	LOFi=LOFj	Conflict	Conflict	Conflict
,B)	LOFi=LOFj O	LOFi=LOFj O	No	LOFi=LOFj O	LOFi=LOFj O	LOFi=LOFj O	Conflict	LOFi=LOFj	Conflict	Conflict
B)	No NC< -> C	Conflict	No ND1 -> C	Conflict	No ND< -> C	Conflict	Conflict	Conflict	No	Conflict
A)	Conflict	No NC> -> C	No ND1 -> C	Conflict	Conflict	No ND> -> C	Conflict	Conflict	Conflict	No

Note: For some cells even though LOFi and LOFj cannot be combined, one of the relationships can be upgraded to a more precise relationship. If this is the case, the upgraded relationship is shown in the second entry.

Table of Transitivity Rules

rel2 / el1	NA	NC<	NC>	ND1	NC2	ND<	ND>	D	O	C
NA	no info no info no info no info	no info no info no info NB(C,A)	NB(A,C) no info no info no info	no info no info	NB(A,C) no info no info NB(C,A)	no info no info no info NA(C,A)	NA(A,C) no info no info no info	NC<(A,C) no info no info NC>(C,A)	NA(A,C) no info no info NA(C,A)	no info NA(A,C) no info NA(C,A)
NC<	NA(A,C) no info no info no info	NC<(A,C) no info no info NC<(C,A)	NB(A,C) no info no info no info	NB(A,C) no info	NC<(A,C) no info no info NC<(C,A)	NC<(A,C) NA(A,C) no info NC<(C,A)	NA(A,C) NA(A,C) no info no info	NC<(A,C) no info no info D(C,A)	NC<(A,C) NA(A,C) no info NC<(C,A)	NC<(A,C) NA(A,C) no info NC<(C,A)
NC>	no info no info no info NA(C,A)	no info no info no info NA(C,A)	NC>(A,C) no info no info NC>(C,A)	no info NA(C,A)	NC>(A,C) no info no info NC>(C,A)	no info no info NA(C,A) NA(C,A)	NC>(A,C) NA(A,C) NA(C,A) NC>(C,A)	D(A,C) no info no info NC>(C,A)	NC>(A,C) NA(C,A) NA(C,A) NC>(C,A)	no info NC>(A,C) NC>(C,A) NA(C,A)
ND1	no info no info	no info NA(C,A)	NB(A,C) no info	no info	NA(A,C) NA(C,A)	no info NA(C,A)	NA(A,C) no info	NC<(A,C) NC>(C,A)	NA(A,C) NA(C,A)	no info ND1(A,C)
NC2	NA(A,C) no info no info NA(C,A)	NC<(A,C) no info no info NC<(C,A)	NC>(A,C) no info no info NC>(C,A)	NB(A,C) NB(C,A)	NC2(A,C) no info no info NC2(C,A)	NC<(A,C) NA(A,C) NA(C,A) NC<(C,A)	NC>(A,C) NA(A,C) NA(C,A) NC>(C,A)	D(A,C) no info no info D(C,A)	NC2(A,C) NA(A,C) NA(C,A) NC2(C,A)	NC<(A,C) NC>(A,C) NC>(C,A) NC<(C,A)
ND<	NA(A,C) no info no info no info	NC<(A,C) NA(C,A) no info NC<(C,A)	NA(A,C) no info NA(A,C) no info	NB(A,C) no info	NC<(A,C) NA(C,A) NA(A,C) NC<(C,A)	NC<(A,C) ND1(A,C) no info NC<(C,A)	NA(A,C) NA(A,C) NA(A,C) no info	NC<(A,C) NC>(C,A) NC<(A,C) D(C,A)	NC<(A,C) ND1(A,C) NA(A,C) NC<(C,A)	NC<(A,C) ND1(A,C) no info ND<(C,A)
ND>	no info no info no info NA(C,A)	no info NA(C,A) no info NA(C,A)	NC>(A,C) no info NA(A,C) NC>(C,A)	no info NB(C,A)	NC>(A,C) NA(C,A) NA(A,C) NC>(C,A)	no info NA(C,A) NA(C,A) NA(C,A)	NC>(A,C) no info ND1(A,C) NC>(C,A)	D(A,C) NC>(C,A) NC<(A,C) NC>(C,A)	NC>(A,C) NA(C,A) ND1(A,C) NC>(C,A)	no info ND>(A,C) NC>(C,A) ND1(A,C)
D	NC>(A,C) no info no info NC<(C,A)	D(A,C) no info no info NC<(C,A)	NC>(A,C) no info no info D(C,A)	NC>(A,C) NC<(C,A)	D(A,C) no info no info D(C,A)	D(A,C) NC>(A,C) NC<(C,A) NC<(C,A)	NC>(A,C) NC>(A,C) NC<(C,A) D(C,A)	D(A,C) no info no info D(C,A)	D(A,C) NC>(A,C) NC<(C,A) D(C,A)	D(A,C) NC>(A,C) D(C,A) NC<(C,A)
O	NA(A,C) no info no info NA(C,A)	NC<(A,C) NA(C,A) no info NC<(C,A)	NC>(A,C) no info NA(A,C) NC>(C,A)	NA(A,C) NA(C,A)	NC2(A,C) NA(C,A) NA(A,C) NC2(C,A)	NC<(A,C) ND1(A,C) NA(C,A) NC<(C,A)	NC>(A,C) NA(A,C) ND1(A,C) NC>(C,A)	D(A,C) NC>(C,A) NC<(A,C) D(C,A)	NC2(A,C) ND1(A,C) ND1(A,C) NC2(C,A)	NC<(A,C) ND>(A,C) NC>(C,A) ND<(C,A)
C	NA(A,C) NA(C,A) no info no info	NC<(A,C) NA(C,A) no info NC<(C,A)	NA(A,C) NC>(C,A) NC>(A,C) no info	ND1(A,C) no info	NC<(A,C) NC>(C,A) NC>(A,C) NC<(C,A)	ND<(A,C) ND1(A,C) no info NC<(C,A)	ND1(A,C) ND>(C,A) NC>(A,C) no info	NC<(A,C) NC>(C,A) D(A,C) D(C,A)	ND<(A,C) ND>(C,A) NC>(A,C) NC<(C,A)	C(A,C) ND1(A,C) no info C(C,A)

In each cell,
ntry 1 = rel1(A,B) and rel2(B,C) entry 3 = rel1(B,A) and rel2(B,C)
ntry 2 = rel1(A,B) and rel2(C,B) entry 4 = rel1(B,A) and rel2(C,B)

D. Table of Evidence-Combination Rules for Transitivity

rel1 [s1,s2,s3] \ rel2 [s4,s5,s6]	NA	NC<	NC>	ND1	NC2	ND<	ND>	D	O	C
NA	–	–	–	–	–	–	–	–	–	–
	–	–	–		–	–	–	–	–	–
	–	–	–	–	–	–	–	–	–	–
	–	–	–		–	–	–	–	–	–
NC<	–	[s1,{},{}]	–	–	[s1,{},{}]	[s1,{},{}]	–	[s1,{},{}]	[s1,{},{}]	[s1,{},{}]
	–	–	–		–	–	–	–	–	–
	–	–	–	–	–	–	–	–	–	–
	–	[s4,{},{}]	–		[s4,{},{}]	[s4,{},{}]	–	[s4,s5Us1,{}]	[s4,{},{}]	[s4,{},{}]
NC>	–	–	[{},{},s6]	–	[{},{},s6]	–	[{},{},s6]	[{},s3Us5,s6]	[{},{},s6]	–
	–	–	–		–	–	–	–	–	[{},{},s6]
	–	–	–	–	–	–	–	–	–	[{},{},s3]
	–	–	[{},{},s3]		[{},{},s3]	–	[{},{},s3]	[{},{},s3]	[{},{},s3]	–
ND1	–	–	–	–	–	–	–	[s2,{},{}]	–	–
	–	–	–		–	–	–	[{},{},s2]	–	[{},s2,{}]
NC2	–	[s1,{},{}]	[{},{},s6]	–	[s1,{},s6]	[s1,{},{}]	[{},{},s6]	[s1,s3Us5,s6]	[s1,{},s6]	[s1,{},{}]
	–	–	–		–	–	–	–	–	[{},{},s6]
	–	–	–	–	–	–	–	–	–	[{},{},s3]
	–	[s4,{},{}]	[{},{},s3]		[s4,{},s3]	[s4,{},{}]	[{},{},s3]	[s4,s5Us1,s3]	[s4,{},s3]	[s4,{},{}]
ND<	–	[s1,{},{}]	–	–	[s1,{},{}]	[s1,{},{}]	–	[s1,{},{}]	[s1,{},{}]	[s1,{},{}]
	–	–	–		–	–	–	[{},{},s2]	–	[{},s2,{}]
	–	–	–	–	–	–	–	[s2,{},{}]	–	–
	–	[s4,{},{}]	–		[s4,{},{}]	[s4,{},{}]	–	[s4,s5Us1,s2]	[s4,{},{}]	[s1Us4,s2,{}]
ND>	–	–	[{},{},s6]	–	[{},{},s6]	–	[{},{},s6]	[s2,s3Us5,s6]	[{},{},s6]	[{},s2,s3Us6]
	–	–	–		–	–	–	[{},{},s2]	–	[{},{},s3]
	–	–	–	–	–	–	–	[s2,{},{}]	–	[{},s2,{}]
	–	–	[{},{},s3]		[{},{},s3]	–	[{},{},s3]	[{},{},s2Us3]	[{},{},s3]	
D	–	[s1,s2Us4,{}]	[{},{},s6]	[{},{},s5]	[s1,s2Us4,s6]	[s1,s2Us4,s5]	[{},{},s5Us6]	[s1,s2Us3Us4Us5,s6]	[s1,s2Us4,s5Us6]	[s1,s2Us4,s5]
	–	–	–		–	[{},{},s5]	[{},{},s5]	–	[{},{},s5]	[{},{},s5Us6]
	–	–	–	[s5,{},{}]	–	[s5,{},{}]	[s5,{},{}]	–	[s5,{},{}]	[s5,s2Us6,s3]
	–	[s4,{},{}]	[{},s2Us6,s3]		[s4,s2Us6,s3]	[s4Us5,{},{}]	[s5,s2Us6,s3]	[s4,s1Us2Us5Us6,s3]	[s4Us5,s2Us6,s3]	[s4Us5,{},{}]
O	–	[s1,{},{}]	[{},{},s6]	–	[s1,{},s6]	[s1,{},{}]	[{},{},s6]	[s1Us2,s3Us5,s6]	[s1,{},s6]	[s1,{},{}]
	–	–	–		–	–	–	[{},{},s2]	–	[{},s2,s3Us6]
	–	–	–	–	–	–	–	[s2,{},{}]	–	[{},{},s3]
	–	[s4,{},{}]	[{},{},s3]		[s4,{},s3]	[s4,{},{}]	[{},{},s3]	[s4,s1Us5,s2Us3]	[s4,{},s3]	[s1Us4,s2,{}]
C	–	[s1,{},{}]	–	[{},s5,{}]	[s1,{},{}]	[s1Us4,s5,{}]	[{},s5,{}]	[s1Us2Us4,{},{}]	[s1Us4,s5,{}]	[s1Us4,s5,s3Us6]
	–	–	[{},{},s3]		[{},{},s3]	[{},s5,{}]	[{},s5,s3Us6]	[{},{},s2Us3Us6]	[{},s5,s3Us6]	[{},s2Us5,{}]
	–	–	[{},{},s6]	–	[{},{},s6]	–	[{},{},s6]	[s2,s3Us5,s6]	[{},{},s6]	–
	–	[s4,{},{}]	–		[s4,{},{}]	[s4,{},{}]	–	[s4,s5Us1,s2]	[s4,{},{}]	[s1Us4,s2,s3Us6]