Image Databases
Image Databases

- In standard relational databases, the user types in a query, and obtains an answer in response.

- In image databases, things are different: for example, a police investigator may have in front of him/her, a surveillance photograph of someone, whose identity s/he may not know, but wishes to determine. Thus, s/he may wish to ask a query of the form: *Here’s a picture of a person. Can you retrieve all pictures from the image database that are “similar” to this person and tell me the identities of the people in the pictures you return to me?*

- This query is fundamentally different from ordinary queries for two reasons:

  1. First, the query includes a picture as part of the query.
  2. Second, the query asks about “similar” pictures and hence, uses a notion of “imprecise match” whose definition needs to be precisely articulated (it is possible to reason precisely about imprecise data!)
Example Face Database
Raw Images

The content of an image consists of all “interesting” objects in that image. Each object is characterized by:

- a *shape descriptor* that describes the shape/location of the region within which the object is located inside a given image.
- a *property descriptor* that describes the properties of the individual pixels (or groups of pixels) in the given image. Examples of such properties include: red-green-blue (RGB) values of the pixel (or aggregated over a group of pixels), grayscale levels in the case of black and white images, etc. In general, it will be infeasible to associate properties with individual pixels, and hence, cells (rectangular “groups” of pixels) will be used most of the time.

- We assume the existence of a set *Prop* of properties. A property consists of two components –
  1. A *property name* – e.g. “Red”, “Green”, “Blue” and
  2. a *property domain* which specifies the range of values that the property can assume – e.g \(\{0, \ldots, 8\}\).
Example

**Example:** Consider the image file `pic1.gif` on the preceding slide. This image has two objects of interest - let call these two objects \( o_1 \) and \( o_2 \).

- The shapes of these objects are captured by rectangles shown. Formally, object \( o_1 \)'s shape may be specified by:

  \[
  \text{rectangle} : XLB = 10; XUB = 60, YLB = 5; YUB = 50.
  \]

- The *property descriptor* associated with an individual cell (group of pixels) may look like this:

  1. Red = 5;
  2. Green = 1;
  3. Blue = 3.

Other properties like texture may also be included.
Definitions

- Every image \( I \) has an associated pair of positive integers \((m, n)\), called the \textit{grid-resolution} of the image. This divides the image into \((m \times n)\) \textit{cells} of equal size, called the \textit{image grid}.

- Each cell in a given gridded \((m \times n)\) image \( I \) consists of a collection of \textit{pixels}.

- A \textit{cell property} is a triple \((\text{Name}, \text{Values}, \text{Method})\) where \text{Name} is a string denoting the property’s name, \text{Values} is a set of values that that property may assume, and \text{Method} is an algorithm that tells us how to compute the property involved.

\textbf{Example:} Consider black and white images.

\[(\text{bwcolor}, \{b, w\}, \text{bwalgo})\]

could be a cell property with property name \text{bwcolor} and the possible values are \(b\) (black) and \(w\) (white), respectively. \text{bwalgo} then is an algorithm which may take a cell as input, and return as output, either black or white, by somehow combining the black/white levels of the pixels in the cell.

\textbf{Example:} Consider gray scale images (with \(0 = \text{white}\) and \(1 = \text{black}\)), we may have the cell property

\[(\text{graylevel}, [0, 1], \text{grayalgo})\]
where our property name is named \texttt{graylevel}, and its possible values are real numbers in the [0, 1] interval, and the associated method \texttt{grayalgo} takes as input, a cell, and computes its gray level.
Image Definitions

- An object shape is any set $P$ of points such that if $p, q \in P$, then there exists a sequence of points $p_1, \ldots, p_n$ all in $P$ such that:

  1. $p = p_1$ and $q = p_n$ and
  2. for all $1 \leq i < n$, $p_{i+1}$ is a neighbor of $p_i$, i.e. if $p_i = (x_i, y_i)$ and $p_{i+1} = (x_{i+1}, y_{i+1})$, then $(x_{i+1}, y_{i+1})$ satisfies one of the following conditions:

    $\begin{align*}
    (x_{i+1}, y_{i+1}) &= (x_i + 1, y_i) & (x_{i+1}, y_{i+1}) &= (x_i - 1, y_i) \\
    (x_{i+1}, y_{i+1}) &= (x_i, y_i + 1) & (x_{i+1}, y_{i+1}) &= (x_i, y_i - 1) \\
    (x_{i+1}, y_{i+1}) &= (x_i + 1, y_i + 1) & (x_{i+1}, y_{i+1}) &= (x_i + 1, y_i - 1) \\
    (x_{i+1}, y_{i+1}) &= (x_i - 1, y_i + 1) & (x_{i+1}, y_{i+1}) &= (x_i - 1, y_i - 1)
    \end{align*}$

- A rectangle is an object shape, $P$, such that there exist integers $XLB, XUB, YLB, YUB$ such that

    $$P = \{ (x, y) \mid XLB \leq x < XUB \& YLB \leq y < YUB \}.$$
Image Database

**Def:** An image database, *IDB*, consists of a triple (*Gl*, *Prop*, *Rec*) where:

1. *Gl* is a set of gridded images of the form (*Image*, *m*, *n*) and
2. *Prop* is a set of cell properties, and
3. *Rec* is a mapping that associates with each image, a set of rectangles denoting objects.
Issues in Image Databases

- First and foremost, images are often very large objects consisting of a \((p_1 \times p_2)\) pixel array. Explicitly storing properties on a pixel by pixel basis is usually infeasible. This has led to a family of image compression algorithms that attempt to compress the image into one containing fewer pixels.

- Given an image \(I\) (compressed or raw), there is a critical need to determine what “features” appear in the image. This is typically done by breaking up the image into a set of homogeneous (w.r.t. some property) rectangular regions, each of which is called a segment. The process of finding these segments is called segmentation.

- Once image data has been segmented, we need to support “match” operations that map either a whole image or a segmented portion of an image against another whole/segmented image.
Compressed Image Representations

- Consider a 2-dimensional image $I$ consisting of $(p_1 \times p_2)$ pixels.
- Let $I(x, y)$ be a number denoting one or more attributes of the pixel.

The creation of the compressed representation, $cr(I)$, of image $I$ consists of two parts:

1. **Size Selection**: The larger the size, the greater is the fidelity of the representation. However, as the size increases, so does the complexity of creating an index for manipulating such representations, and searching this index. Let $cr(I)$ be of size $h_1 \times h_2$ where $h_i \leq p_i$.

2. **Transform Selection**: The user must select a transformation, which, given the image $I$, and any pairs of number $1 \leq i \leq h_1$, and $1 \leq j \leq h_2$ will determine what the value of $cr(i, j)$ is.

3. There are many such transforms.
The Discrete Fourier Transform (DFT)

\[
\text{DFT}(i, j) = \frac{1}{\sqrt{p_1}} \times \frac{1}{\sqrt{p_2}} \times \\
\sum_{a=0}^{p_1} \sum_{b=0}^{p_2} \left( I(a, b) \times \exp \left( -\frac{2\pi ja \times i}{p_1} \right) \times \frac{-2\pi jb \times i}{p_2} \right).
\]

where: \( j \) is the well known complex number, \( \sqrt{-1} \).

DFT has many nice properties.

- **Invertibility:** It is possible to “get back” the original image \( I \) from its DFT representation. Useful for decompression.

- **Note that practical realizations of DFT ave often sacrificed this property by applying the DFT together with certain other non-invertible operations.**

- **Distance Preservation:** DFT preserves Euclidean distance. This is important in image matching applications where we may wish to use distance measures to represent similarity levels.
The Discrete Cosine Transform

\[ \text{DCT}(i, j) = \frac{2}{\sqrt{p_1 \times p_2}} \alpha(i) \times \alpha(j) \]

\[ \sum_{r=0}^{p_1-1} \sum_{s=0}^{p_2-1} \cos \left( \frac{(2r + 1) \times \pi i}{2r} \right) \times \cos \left( \frac{(2s + 1) \times \pi j}{2s} \right) \]

where:

\[ \alpha(i), \alpha(j) = \begin{cases} \frac{1}{\sqrt{2}} & \text{when } u, v = 0 \\ 1 & \text{otherwise.} \end{cases} \]

- DCT also is easily invertible
- DCT can be computed quite fast.

Other compression techniques include Wavelets.
Image Processing: Segmentation

- This is the process of taking as input, an image, and producing as output, a way of “cutting up” the image into disjoint regions such that each region is “homogeneous”.

- Suppose $I$ is an image containing $(m \times n)$ cells.

- A connected region, $R$, in image $I$, is a set of cells such that if cells $(x_1, y_1), (x_2, y_2) \in R$, there there exists a sequence of cells $C_1, \ldots, C_n$ in $R$ such that:
  
  1. $C_1 = (x_1, y_1)$ and
  2. $C_n = (x_2, y_2)$ and
  3. The Euclidean distance between cells $C_i$ and $C_{i+1}$ for all $i < n$ is 1.
• Each of $R_1$, $R_2$, $R_3$ is a connected region.

• $(R_1 \cup R_2)$ is a connected region;

• $(R_2 \cup R_3)$ is a connected region;

• $(R_1 \cup R_2 \cup R_3)$ is a connected region;

• But $(R_1 \cup R_3)$ is not a connected region. The reason for this is that the Euclidean distance between the cell $(2, 3)$ which represents the rightmost of the two cells of $R_1$ and the cell $(3, 4)$ which represents the only cell of $R_3$ is $\sqrt{2} > 1$. 
Homogeneity Predicates

- A homogeneity predicate associated with an image $I$ is a function $H$ that takes as input, any connected region $R$ in image $I$, and returns either “true” or “false.”

- **Example:** Suppose $\delta$ is some real number between 0 and 1, inclusive, and we are considering black and white images. We may define a simple homogeneity predicate, $H^{bw}_\delta$ as follows: $H^{bw}_\delta(R)$ returns “true” if over $(100*\delta)\%$ of the cells in region $R$ have the same color.

  - Consider three regions now, as described in the following table:

    | Region | Num. of Black Pixels | Num. of White Pixels |
    |--------|----------------------|----------------------|
    | $R_1$  | 800                  | 200                  |
    | $R_2$  | 900                  | 100                  |
    | $R_3$  | 100                  | 900                  |

  - Suppose we consider some different predicates, $H^{bw}_{0.8}$, $H^{bw}_{0.89}$ and $H^{bw}_{0.92}$.

  - The following table shows us the results returned by these three homogeneity predicates on the above table $R$.

    | Region | $H^{bw}_{0.8}$ | $H^{bw}_{0.89}$ | $H^{bw}_{0.92}$ |
    |--------|----------------|----------------|----------------|
    | $R_1$  | true           | false          | false          |
    | $R_2$  | true           | true           | false          |
    | $R_3$  | true           | true           | false          |
Another Homogeneity Predicate Example

- Suppose each pixel has a real value between 0 and 1, inclusive.
- This value is called the bw-level.
- 0 denotes “white”, 1 denotes “black”, and everything in between denotes a shade somewhere between black and white.
- Suppose $f$ assigns numbers between 0 and 1 (inclusive) to each cell. In addition, you have a “noise factor” $0 \leq \eta \leq 1$, and a threshold $\delta$ as in the preceding case.
- $H^{f,\eta,\delta}(R)$ is now “true” iff
  \[
  \left\{ (x, y) \mid \frac{|\text{bwlevel}(x, y) - f(x, y)| < \eta}{(m \times n)} > \delta \right\} > \delta.
  \]
- What this homogeneity predicate does is to use a “baseline” function $f$, and a maximal permissible noise level $\eta$. It considers the bw-level of cell $(x, y)$ to be sufficiently similar to that predicted by $f$ if
  \[|\text{bwlevel}(x, y) - f(x, y)| < \eta,\]
i.e. if the two differ by no more than $\eta$.
- It then checks to see if sufficiently many cells (which is determined by the factor $\delta$) in the region “match” the predictions made by $f$. If so, it considers the region $R$ to be homogeneous, and returns “true.” Otherwise, it returns “false.”
Segmentation

**Def:** Given an image $I$ represented as a set of $(m \times n)$ pixels, we define a *segmentation* of image $I$ w.r.t. a homogeneity predicate $P$ to be a set $R_1, \ldots, R_k$ of regions such that:

1. $R_i \cap R_j = \emptyset$ for all $1 \leq i \neq j \leq k$ and
2. $I = R_1 \cup \cdots \cup R_k$;
3. $H(R_i) =$ “true” for all $1 \leq i \leq k$;
4. For all distinct $i, j$, $1 \leq i, j \leq n$ such that $R_i \cup R_j$ is a connected region, it is the case that $H(R_i \cup R_j) =$ “false.”

**Example:** For example, consider a simple $(4 \times 4)$ region containing the bw-levels shown in the table below.

<table>
<thead>
<tr>
<th>Row/Col</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1</td>
<td>0.25</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>0.05</td>
<td>0.30</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>3</td>
<td>0.35</td>
<td>0.30</td>
<td>0.55</td>
<td>0.8</td>
</tr>
<tr>
<td>4</td>
<td>0.6</td>
<td>0.63</td>
<td>0.85</td>
<td>0.90</td>
</tr>
</tbody>
</table>

Consider now, the homogeneity predicate $H_{1}^{dyn,0.03}$. This homogeneity predicate says that a region $R$ is to be considered homogeneous iff there exists an $r$ such that each and every cell in the
region has a bw-level $v$ such that

$$|v - r| \leq 0.03.$$  

According to this classification, it is easy to see that the following five regions constitute a valid segmentation of the above image w.r.t. $H^{dyn,0.03}_1$.

$$R_1 = \{(1,1), (1,2)\}.$$  
$$R_2 = \{(1,3), (2,1), (2,2), (2,3)\}.$$  
$$R_3 = \{(3,1), (3,2), (3,3), (4,1), (4,2)\}.$$  
$$R_4 = \{(3,4), (4,3), (4,4)\}.$$  
$$R_5 = \{(1,4), (2,4)\}.$$
Segmentation Algorithm Sketch

- **Split**: In this method, we start with the whole image. If it is homogeneous, then we are done, and the image is a valid segmentation of itself. Otherwise, we split the image into two parts, and recursively repeat this process, till we find a set $R_1, \ldots, R_n$ of regions that are homogeneous, and satisfy all conditions, except the fourth condition in the definition of “homogeneity” predicates.

- **Merge**: We now check which of the $R_i$’s can be merged together. At the end of this step, we will obtain a valid segmentation $R'_1, \ldots, R'_k$ of the image, where $k \leq n$ and where each $R'_i$ is the union of some of the $R_j$’s.
Segmentation Algorithm

function segment(I:image);
    SOL = ∅;
    check_split(I);
    merge(SOL);
end function

function check_split(R);
    if $H(R) = \text{'true'}$ then addsol(R)
    else
        \{ X = split(R);
        check_split(X.part1);
        check_split(X.part2);
        \}
end function

procedure addsol(R);
    SOL = SOL ∪ \{R\}
end procedure
function merge(S);
    while $S \neq \emptyset$ do {
        Pick some $Cand$ in $S$;
        merged = false;
        $S = S - \{Cand\}$;
        Enumerate $S$ as $C_1, \ldots, C_k$;
        while $i \leq k$ do
            { if adjacent($Cand, C_i$) then
                { Cand = Cand $\cup$ $C_i$;
                    $S = S - \{C_i\}$;
                    merged = true;
                }
            } else {$i = i + 1$;
                if merged then $S = S \cup \{Cand\}$;
                merged = false
            }
        }
    }
end function
Similarity Based Retrieval

Which of these images is similar to the other?

(a) One Monkey (chimp)       (b) Another Monkey (orangutan)

(a) One Shark (tiger)         (b) Another Shark (gray reef)
Similarity Based Retrieval

Two approaches:

- **(The Metric Approach)** Assume there is a distance metric $d$ that can compare any two image objects. The closer two objects are in distance, the more similar they are considered to be. Given an input image $i$, find the “nearest” neighbor of $i$ in the image archive. This is the most widely followed approach in the database world.

- **(The Transformation Approach)** The metric approach assumes that the notion of similarity is “fixed”, i.e. in any given application only one notion of similarity is used to index the data (though different applications may use different notions of similarity). Computes the “cost” of transforming one image into another based on user-specified cost functions that may vary from one query to another.
Metric Approach

- Suppose we consider a set Obj of objects, having pixel properties $p_1, \ldots, p_n$, as described earlier in this chapter. Thus, each object $o$ may be viewed as a set $S(o)$.

- A function $d$ from some set $X$ to the unit interval $[0, 1]$ is said to be a distance function if it satisfies the following axioms for all $x, y, z \in X$:

\[
\begin{align*}
    d(x, y) &= d(y, x), \\
    d(x, z) &\leq d(x, z) + d(z, y), \\
    d(x, x) &= 0.
\end{align*}
\]

- Let $d_{Obj}$ be a distance function on the space of all objects in our domain, i.e. $d_{Obj}$ is a distance function on a $k = (n + 2)$ dimensional space.

- **Example**: Obj to consist of $(256 \times 256)$ images having three attributes (red, green, blue) each of which assumes a value from the set $\{0, \ldots, 7\}$. Could have:

\[
\begin{align*}
    d_i(o_1, o_2) &= \sqrt{\sum_{i=1}^{256} \sum_{j=1}^{256} \left( \text{diff}_{r}[i, j] + \text{diff}_{g}[i, j] + \text{diff}_{b}[i, j] \right)} \\
    \text{diff}_{r}[i, j] &= (o_1[i, j].\text{red} - o_2[i, j].\text{red})^2 \\
    \text{diff}_{g}[i, j] &= (o_1[i, j].\text{green} - o_1[i, j].\text{green})^2 \\
    \text{diff}_{b}[i, j] &= (o_1[i, j].\text{blue} - o_1[i, j].\text{blue})^2
\end{align*}
\]
• Such computations can be cumbersome because the double summation leads to 65536 expressions being computed inside the sum.
Metric Approach

- How can this massive similarity computation be avoided?
- Suppose we have a “good” feature extraction function $\mathbf{fe}$.
- Use $\mathbf{fe}$ to map objects into single points in a $s$-dimensional space, where $s$ would typically be pretty small compared to $(n + 2)$.
- This leads to two reductions:
  1. First, recall that an object $o$ is a set of points in an $(n + 2)$ dimensional space. In contrast, $\mathbf{fe}(o)$ is a single point.
  2. Second, $\mathbf{fe}(o)$ is a point in an $s$-dimensional space where $s \ll (n + 2)$.
- Mapping must preserve distance, i.e. if $o_1, o_2, o_3$ are objects such that the distance $d(o_1, o_2) \leq d(o_1, o_3)$, then $d'(\mathbf{fe}(o_1), \mathbf{fe}(o_2)) \leq d'(\mathbf{fe}(o_1), \mathbf{fe}(o_3))$ where $d$ is a metric on the original $(n + 2)$ dimensional space, and $d'$ is a metric on the new, $s$-dimensional space. In other words, the feature extraction map should preserve the distance relationships in the original space.
Reducing Dimensionality of Feature Space

(n+2) dim. space  Map FE  s-dimensional space  Indexing Algorithm  INDEX  Object Repository
Index Creation Algorithm

Input: \( Obj \), a set of objects.

1. \( T = NIL \). \(* T \) is an empty quadtree, or R-tree for \( s \)-dimensional data \(*\)

2. \textbf{if} \( Obj = \emptyset \) \textbf{then return} \( T \) and \textbf{halt}.

3. \textbf{else}
   
   (a) Compute \( \text{fe}(o) \),
   
   (b) Insert \( \text{fe}(o) \) into \( T \).
   
   (c) \( Obj = Obj - \{o\} \).
   
   (d) Goto 2
Finding the Best Matches

FindMostSimilarObject Algorithm

Input: a tree $T$ of the above type. An object $o$.

1. bestnode = NIL;
2. if $T = NIL$ then return bestnode. Halt
3. else

- find the nearest neighbors of $fe(o)$ in $T$ using a nearest neighbor search technique. If multiple such neighbors exist, return them all.
Finding “Sufficiently” Similar Objects

FindSimilarObjects Algorithm

Input: a tree $T$ of the above type. An object $o$. A tolerance $0 < \epsilon \leq 1$.

1. Execute a range query on tree $T$ with center $f(e)(o)$ and radius $\epsilon$.
2. Let $o_1, \ldots, p_r$ be all the points returned.
3. for $i = 1$ to $r$ do
   (a) if $d(o, f^{-1}(o_i)) \leq \epsilon$ then print $f^{-1}(o_i)$.

The above algorithm works only if the distance metric in the space of small dimensionality (i.e. dimension $s$) consistently overestimates the distance metric $d$. 
The Transformation Approach

- Based on the principle that given two objects \( o_1, o_2 \), the level of dis-similarity between \( o_1, o_2 \) is proportional to the (minimum) cost of transforming object \( o_1 \) into object \( o_2 \), or vice-versa.

- We start with a set of transformation operators, \( t_{o_1}, \ldots, t_{o_r} \), e.g.
  
  - translation
  - rotation
  - scaling – uniform and nonuniform
  - excision (that culls out a part of an image)

- The transformation of object \( o \) into object \( o' \) is a sequence of transformation operations \( t_{o_1}, \ldots, t_{o_r} \) and a sequence of objects \( o_1, \ldots, o_r \) such that:
  
  1. \( t_{o_1}(o) = o_1 \) and
  2. \( t_{o_i}(o_{i-1}) = o_i \) and
  3. \( t_{o_r}(o_r) = o' \).

The cost of the above transformation sequence, \( TS \) is given by:

\[
\text{cost}(TS) = \sum_{i=1}^{r} \text{cost}(t_{o_i}).
\]
The Transformation Approach

- Suppose $\text{TSeq}(o, o')$ is the set of all transformation sequences that convert $o$ into $o'$.

- The dissimilarity between $o$ and $o'$, denoted $\text{dis}(o, o')$ w.r.t. a set $TR$ of transformation operators, and a set $CF$ of cost functions is given by:

$$
\text{dis}(o, o') = \min \{ \text{cost}(TS) \mid TS \in \text{TSeq}(o, o') \cup \text{TSeq}(o', o) \}.
$$
Example

$TS_1$: This transformation sequence consists of a

- non-uniform scaling operation (scale the blue part of $o_1$ 50% in the vertical upward direction, leaving the horizontal unchanged),

- a non-uniform scaling operation (scale the green part of object $o_1$ by a 100% increase in the vertical, downward direction, with no change in the horizontal).

- The third operation applies the **paint** operation, painting the two pixels colored magenta to green.

- The Figure below depicts the intermediate steps.
(a) non-uniform scaling

(b) paint
Example

$TS_2$: This transformation sequence consists of a

- non-uniform scaling operation (scale the blue part of $o_1$ 50% in the vertical upward direction, leaving the horizontal unchanged).

- apply the `paint` operation, painting the green object magenta.

- apply the non-uniform scaling operation (scale the magenta part of object $o_1$ by a 100% increase in the vertical, downward direction, with no change in the horizontal).

- The following figure depicts the intermediate steps.
• If we assume that the cost functions associated with non-uniform scaling are independent of color, and the paint operation merely counts the number of pixels being painted, then it is easy to see that transformation $TS_2$ accomplishes the desired transformation at a cheaper cost (as it paints one less pixel than transformation $TS_1$.)
Transformation Model vs. the Metric Model

Advantages of the Transformation Model over the Metric Model

- First and foremost, the user can “set up” his own notion of similarity by specifying that certain transformation operators may/may not be used.

- Second, the user may associate, with each transformation operator, a cost function that assesses a cost to each application of the operation, depending upon the arguments to the transformation operator. This allows the user to personalize the notion of similarity for his/her needs.

Advantages of the Metric Model over the Transformation Model

By forcing the user to use one and only one (dis)similarity metric, the system can facilitate the indexing of data so as to optimize the one operation of finding the “nearest” neighbor (i.e. least dis-similar) object w.r.t. the query object specified by the user.
Alternative Image DB Paradigms

- Representing IDBs as Relations
- Representing IDBs with Spatial Data Structures
- Representing IDBs using Image Transformations

Two different photographs of the same person may vary, depending upon a variety of factors such as:

1. the time of the day at which the two photographs were taken;
2. the lighting conditions under which the photographs were taken;
3. the camera used;
4. the exact position of the subject’s head and his/her facial expression.
5. etc.
Representing Image DBs with Relations

Suppose \( \text{IDB} = (\text{Gl, Prop, Rec}). \)

1. Create a relation called \text{images} having the scheme:

\[
(\text{Image, ObjID, XLB, XUB, YLB, YUB})
\]

where \text{Image} is the name of an image file, \text{ObjID} is a dummy name created for an object contained in the image, and \text{XLB, XUB, YLB, YUB} describe the rectangle in question. If \( R \) is a rectangle specified by \text{XLB, XUB, YLB, YUB} and \( R \) is in \( \text{Rec}(I) \), then there exists a tuple

\[
(I, \text{newid}, \text{XLB, XUB, YLB, YUB})
\]

in the relation \text{images}.

2. For each property \( p \in \text{Prop} \), create a relation \( R_p \) having the scheme:

\[
(\text{Image, XLB, XUB, YLB, YUB, Value}).
\]

Here, \text{Image} is the name of an image file. Unlike the preceding case though, \text{XLB, XUB, YLB, YUB} denote a rectangular \text{cell} in the image, and \text{Value} specifies the value of the property \( p \).

Example:
<table>
<thead>
<tr>
<th>Image</th>
<th>ObjId</th>
<th>XLB</th>
<th>XUB</th>
<th>YLB</th>
<th>YUB</th>
</tr>
</thead>
<tbody>
<tr>
<td>pic1.gif</td>
<td>$o_1$</td>
<td>10</td>
<td>60</td>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>pic1.gif</td>
<td>$o_2$</td>
<td>80</td>
<td>120</td>
<td>20</td>
<td>55</td>
</tr>
<tr>
<td>pic2.gif</td>
<td>$o_3$</td>
<td>20</td>
<td>65</td>
<td>20</td>
<td>75</td>
</tr>
<tr>
<td>pic3.gif</td>
<td>$o_4$</td>
<td>25</td>
<td>75</td>
<td>10</td>
<td>60</td>
</tr>
<tr>
<td>pic4.gif</td>
<td>$o_5$</td>
<td>20</td>
<td>60</td>
<td>30</td>
<td>80</td>
</tr>
<tr>
<td>pic5.gif</td>
<td>$o_6$</td>
<td>0</td>
<td>40</td>
<td>15</td>
<td>50</td>
</tr>
<tr>
<td>pic6.gif</td>
<td>$o_7$</td>
<td>20</td>
<td>75</td>
<td>15</td>
<td>80</td>
</tr>
<tr>
<td>pic6.gif</td>
<td>$o_8$</td>
<td>20</td>
<td>70</td>
<td>130</td>
<td>185</td>
</tr>
<tr>
<td>pic7.gif</td>
<td>$o_9$</td>
<td>15</td>
<td>70</td>
<td>15</td>
<td>75</td>
</tr>
</tbody>
</table>
Image Properties

- Pixel Level Properties: e.g. RGB values
- Object/Region Level Properties: e.g. NAME, AGE
- Image Level Properties: e.g. when image was captured, where, and by whom
Querying (Relational Representations of) Image DBs

- Eliciting the contents of an image is done using image processing algorithms.
- Image processing algorithms are usually only partially accurate.
- This implies that tuples placed in a relation by an image processing program has certain associated probabilistic attributes.
- Each has object has a value associated with each property $p \in \text{Prop}$.
- Each relation $R_p$ associated with object/region level properties as well as image level properties has just two attributes: an object (or region) id, and a value for the property.
- Example:

  Relation name
\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
ObjId & Name \\
\hline
\textit{o_1} & Jim Hatch \\
\hline
\textit{o_2} & John Lee \\
\hline
\textit{o_3} & John Lee \\
\hline
\textit{o_4} & Jim Hatch \\
\hline
\textit{o_5} & Bill Bosco \\
\hline
\textit{o_6} & Dave Dashell \\
\hline
\textit{o_7} & Ken Yip \\
\hline
\textit{o_8} & Bill Bosco \\
\hline
\textit{o_9} & Ken Yip \\
\hline
\end{tabular}
\end{table}

- This description must be enhanced to include probabilistic object identification.
Probabilistic Version of the name relation

<table>
<thead>
<tr>
<th>ObjId</th>
<th>Name</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>o_1</td>
<td>Jim Hatch</td>
<td>0.8</td>
</tr>
<tr>
<td>o_1</td>
<td>Dave Fox</td>
<td>0.2</td>
</tr>
<tr>
<td>o_2</td>
<td>John Lee</td>
<td>0.75</td>
</tr>
<tr>
<td>o_2</td>
<td>Ken Yip</td>
<td>0.15</td>
</tr>
<tr>
<td>o_3</td>
<td>John Lee</td>
<td>1</td>
</tr>
<tr>
<td>o_4</td>
<td>Jim Hatch</td>
<td>1</td>
</tr>
<tr>
<td>o_5</td>
<td>Bill Bosco</td>
<td>1</td>
</tr>
<tr>
<td>o_6</td>
<td>Dave Dashell</td>
<td>1</td>
</tr>
<tr>
<td>o_7</td>
<td>Ken Yip</td>
<td>0.7</td>
</tr>
<tr>
<td>o_7</td>
<td>John Lee</td>
<td>0.3</td>
</tr>
<tr>
<td>o_8</td>
<td>Bill Bosco</td>
<td>0.6</td>
</tr>
<tr>
<td>o_8</td>
<td>Dave Dashell</td>
<td>0.2</td>
</tr>
<tr>
<td>o_8</td>
<td>Jim Hatch</td>
<td>0.10</td>
</tr>
<tr>
<td>o_9</td>
<td>Ken Yip</td>
<td>1</td>
</tr>
</tbody>
</table>

Reading:

1. The probability that “John Lee” is the name attribute of \( o_2 \) is 0.75.
2. The probability that “Ken Yip” is the name attribute of \( o_2 \) is 0.15.
3. There is, in this case, a 10% missing probability.
Complex Queries

- Suppose we ask the query *What is the probability that* `pic1.gif` *contains both Jim Hatch and Ken Yip?* Is the answer the product of the two probabilities, i.e. is it $(0.8 \times 0.15) = 0.12$?

- *What is the probability that* `pic6.gif` *contains both Jim Hatch and Ken Yip?* Is the answer $(0.7 \times 0.1) = 0.07$?

- In general, the answer is NO.

**Example:** Consider a hypothetical image `pic8.gif` with two objects $o_{10}, o_{11}$ in it, and suppose our table above is expanded by the insertion of the following new tuples identified by the image processing algorithm.

<table>
<thead>
<tr>
<th>ObjId</th>
<th>Name</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$o_{10}$</td>
<td>Ken Yip</td>
<td>0.5</td>
</tr>
<tr>
<td>$o_{10}$</td>
<td>Jim Hatch</td>
<td>0.4</td>
</tr>
<tr>
<td>$o_{11}$</td>
<td>Jim Hatch</td>
<td>0.8</td>
</tr>
<tr>
<td>$o_{11}$</td>
<td>John Lee</td>
<td>0.1</td>
</tr>
</tbody>
</table>

If we are ignorant about the dependencies between different events (as we are in the above case, then we are forced to confront *four possibilities*:
Possibility 1 $o_{10}$ is Ken Yip and $o_{11}$ is John Hatch.

Possibility 2 $o_{10}$ is Ken Yip and $o_{11}$ is \textit{not} John Hatch.

Possibility 3 $o_{10}$ is \textit{not} Ken Yip but $o_{11}$ is John Hatch.

Possibility 4 $o_{10}$ is \textit{not} Ken Yip and $o_{11}$ is \textit{not} John Hatch.
Complex Queries

- Suppose $p_i$ denotes the probability of Possibility $i$, $1 \leq i \leq 4$. Then, we can say that:

\[
\begin{align*}
    p_1 + p_2 &= 0.5. \\
    p_3 + p_4 &= 0.5. \\
    p_1 + p_3 &= 0.8. \\
    p_2 + p_4 &= 0.2 \\
    p_1 + p_2 + p_3 + p_4 &= 1.
\end{align*}
\]

- The first equation follows from the fact that $o_{10}$ is Ken Yip according to possibilities 1 and 2, and we know from the table, that the probability of $o_{10}$ being Ken Yip is 0.5.

- The second equation follows from the fact that $o_{10}$ is someone other than Ken Yip according to possibilities 3 and 4, and we know from the table, that the probability of $o_{10}$ not being Ken Yip is 0.5.

- The third equation follows from the fact that $o_{11}$ is Jim Hatch according to possibilities 1 and 3, and we know from the table, that the probability of $o_{11}$ being Jim Hatch is 0.8.

- Finally, the last equation follows from the fact that $o_{11}$ is someone other than Jim Hatch according to possibilities 2 and 4, and we know from the table, that the probability of $o_{11}$ not being John Hatch is 0.2.
• In order to determine the probability that pic8.gif contains both Ken Yip and John Hatch, we must attempt to solve the above system of linear equations for $p_1$, keeping in mind the fact that all scenarios possible are covered by our four possibilities. The result we obtain, using a linear programming engine, is that $p_1$'s probability is not uniquely determinable. It could be as low as 0.3 or as high as 0.5, or anywhere in between. In particular, note that merely multiplying the probability of 0.5 associated with Ken Yip being object $o_{10}$ and the probability value 0.8 of m Hatch being object $o_{11}$ leads to a probability of 0.4 which is certainly inside this interval, but does not accurately capture the four possibilities listed above.

• Requires the use of interval probabilities.
Interval Probability Model

Interval Probabilistic Version of the name relation with pic8.gif included

<table>
<thead>
<tr>
<th>ObjId</th>
<th>Name</th>
<th>Prob (Lower)</th>
<th>Prob (Upper)</th>
</tr>
</thead>
<tbody>
<tr>
<td>o₁</td>
<td>Jim Hatch</td>
<td>0.77</td>
<td>0.83</td>
</tr>
<tr>
<td>o₁</td>
<td>Dave Fox</td>
<td>0.17</td>
<td>0.23</td>
</tr>
<tr>
<td>o₂</td>
<td>John Lee</td>
<td>0.72</td>
<td>0.78</td>
</tr>
<tr>
<td>o₂</td>
<td>Ken Yip</td>
<td>0.12</td>
<td>0.18</td>
</tr>
<tr>
<td>o₃</td>
<td>John Lee</td>
<td>0.97</td>
<td>1.00</td>
</tr>
<tr>
<td>o₄</td>
<td>Jim Hatch</td>
<td>0.97</td>
<td>1.00</td>
</tr>
<tr>
<td>o₅</td>
<td>Bill Bosco</td>
<td>0.97</td>
<td>1.00</td>
</tr>
<tr>
<td>o₆</td>
<td>Dave Dashell</td>
<td>0.97</td>
<td>1.00</td>
</tr>
<tr>
<td>o₇</td>
<td>Ken Yip</td>
<td>0.67</td>
<td>0.73</td>
</tr>
<tr>
<td>o₇</td>
<td>John Lee</td>
<td>0.27</td>
<td>0.33</td>
</tr>
<tr>
<td>o₈</td>
<td>Bill Bosco</td>
<td>0.57</td>
<td>0.63</td>
</tr>
<tr>
<td>o₈</td>
<td>Dave Dashell</td>
<td>0.17</td>
<td>0.23</td>
</tr>
<tr>
<td>o₈</td>
<td>Jim Hatch</td>
<td>0.07</td>
<td>0.13</td>
</tr>
<tr>
<td>o₉</td>
<td>Ken Yip</td>
<td>0.97</td>
<td>1.00</td>
</tr>
<tr>
<td>o₁₀</td>
<td>Ken Yip</td>
<td>0.47</td>
<td>0.53</td>
</tr>
<tr>
<td>o₁₀</td>
<td>Jim Hatch</td>
<td>0.37</td>
<td>0.43</td>
</tr>
<tr>
<td>o₁₁</td>
<td>Jim Hatch</td>
<td>0.77</td>
<td>0.83</td>
</tr>
<tr>
<td>o₁₀</td>
<td>Hohn Lee</td>
<td>0.07</td>
<td>0.13</td>
</tr>
</tbody>
</table>

- "Find an image that contains both Ken Yip and Jim Hatch."
Ch. 5  Image Databases

• Let us re-examine the image \texttt{pic8.gif} and see what the probability of this image containing both Ken Yip and Jim Hatch is.

• Constraints generated:

\[
0.47 \leq p_1 + p_2 \leq 0.53.
\]

\[
0.47 \leq p_3 + p_4 \leq 0.53.
\]

\[
0.77 \leq p_1 + p_3 \leq 0.83.
\]

\[
0.17 \leq p_2 + p_4 \leq 0.23.
\]

\[
p_1 + p_2 + p_3 + p_4 = 1.
\]

• Solving the above linear program for minimal and maximal values of the variable \(p_1\), we obtain 0.24 and 0.53, respectively.

• Beauty is that when implementing this, we can avoid solving the linear program altogether.
A General Approach

- A probabilistic relation over a scheme \((A_1, \ldots, A_n)\) is an ordinary relation over the scheme \((A_1, \ldots, A_n, LB, UB)\) where the domain of the \(LB\) and \(UB\) attributes is the unit interval \([0, 1]\) of real numbers.

- In particular, the relation \textit{name} is a probabilistic relation that has three attributes:

\[
(ImageId, ObjectId, Name)
\]

of the sort we have already seen thus far.

- The \textit{name} relation satisfies some integrity constraints:

\[
(\forall t_1, t_2) t_1.\text{ObjId} = t_2.\text{ObjId} \rightarrow t_1.\text{ImageId} = t_2.\text{ImageId}.
\]

This constraint states that an ObjectId can be associated with only one image, i.e. distinct images have distinct ObjectIds. The following constraint says that the LB field of any tuple is always smaller than the UB field.

\[
(\forall t) t.LB \leq t.UB.
\]

- An image database consists of a probabilistic relation called \textit{name} of the above form, together with a set of \textit{ordinary} (i.e. non-probabilistic) relations \(R_1, \ldots, R_k\) corresponding to image properties.
Membership Queries

- A **membership query** in an image database is a query of the form: Find all images in the image database that contain objects named \( s_1, \ldots, s_n \).

- **SELECT**  
  `ImageId`
  **FROM**  
  `name T_1, \ldots, T_n`
  **WHERE**  
  \( T_1.Name = s_1 \) AND \( \cdots \) AND \( T_n.Name = s_n \) AND  
  \( T_1.ImageId = T_2.ImageId \) AND \( \cdots \) AND  
  \( T_1.ImageId = T_n.ImageId. \)

The result of this membership query is a table containing three fields – the `ImageId` fields that is explicitly listed in the query, a LB field, and a UB field. \((im, \ell, u)\) is in the result iff for each \( 1 \leq j \leq n \), there exists a tuple \( t_j \in \text{name} \) such that:

1. \( t.ImageId = im \) and
2. \( t.LB = \ell_i \) and \( t.UB = u_i \) and
3. \( [\ell, u] = [\ell_1, u_1] \otimes [\ell_2, u_2] \otimes \cdots \otimes [\ell_n, u_n] \)

where

\[
[x, y] \otimes [x', y'] = [\max(0, x + x' - 1), \min(y, y')].
\]
Other Queries

- Find all people who have had deposits of over 9000 dollars, and who have been photographed with Denis Jones.

SELECT I.ImageId
FROM name I, bank B
WHERE I CONTAINS B.name, Denis Jones AND B.trans=deposit AND B.amount > 9000 AND B.name = I.name.
1. Create a relation called **occurs in** with two attributes

   \[(\text{ImageId, ObjId})\]

   specifying which objects appear in which images.

2. Create *one* R-tree that stores all the rectangles. If the same rectangle (say with \(XLB = 5, XUB = 15, YLB = 20, YUB = 30\)) appears in two images, then we have an overflow list associated with that node in the R-tree.

3. Each rectangle has an associated set of fields that specifies the object/region level properties of that rectangle.
Example
occursin Relation

<table>
<thead>
<tr>
<th>pic1.gif</th>
<th>$o_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>pic1.gif</td>
<td>$o_2$</td>
</tr>
<tr>
<td>pic2.gif</td>
<td>$o_3$</td>
</tr>
<tr>
<td>pic3.gif</td>
<td>$o_4$</td>
</tr>
<tr>
<td>pic4.gif</td>
<td>$o_5$</td>
</tr>
<tr>
<td>pic5.gif</td>
<td>$o_6$</td>
</tr>
<tr>
<td>pic6.gif</td>
<td>$o_7$</td>
</tr>
<tr>
<td>pic6.gif</td>
<td>$o_8$</td>
</tr>
<tr>
<td>pic7.gif</td>
<td>$o_9$</td>
</tr>
</tbody>
</table>
R-tree representation

facenode = record
  \( R_{e_1}, R_{e_2}, R_{e_3} \): rectangle;
  \( P_1, P_2, P_3 \): \text{\textup{rtnodetype}}
end

rectangle = record
  \( X_{LB}, X_{UB}, Y_{LB}, Y_{UB} \): \text{integer};
  objlist: \text{\textup{objnode}};
  day, mth, yr: \text{\textup{integer}};
  camera_type: \text{\textup{string}};
  place: \text{\textup{string}}
end

objnode = record
  objid: \text{\textup{string}};
  imageid: \text{\textup{string}};
  info: \text{\textup{infotype}}
end

infotype = record
  objname: \text{\textup{string}};
  Lp, Up: \text{\textup{real}}: (* lower and upper probability bounds *)
  Next: \text{\textup{objinfo}}
end
R-Tree Construction

- Get Rectangles
- Create R-tree
- Flesh out Objects
Get Rectangles

First and foremost, we may construct a small table describing all the rectangles that occur, and the images they occur in.

<table>
<thead>
<tr>
<th>ObjId</th>
<th>ImageId</th>
<th>XLB</th>
<th>XUB</th>
<th>YLB</th>
<th>YUB</th>
</tr>
</thead>
<tbody>
<tr>
<td>$o_1$</td>
<td>pic1.gif</td>
<td>10</td>
<td>60</td>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>$o_2$</td>
<td>pic1.gif</td>
<td>80</td>
<td>120</td>
<td>20</td>
<td>55</td>
</tr>
<tr>
<td>$o_3$</td>
<td>pic2.gif</td>
<td>20</td>
<td>65</td>
<td>20</td>
<td>75</td>
</tr>
<tr>
<td>$o_4$</td>
<td>pic3.gif</td>
<td>25</td>
<td>75</td>
<td>10</td>
<td>60</td>
</tr>
</tbody>
</table>
Create R-Tree

We then create an R-tree representing the above rectangles. At this stage the object nodes in the R-tree may still not be “filled in” completely.
Flesh Out Objects

We then “flesh out” and appropriately fill in the fields of the various objects stored in the R-tree.

LEGEND:

<table>
<thead>
<tr>
<th>objid</th>
<th>imageid</th>
<th>nxt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>info</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>objname</th>
<th>Lp</th>
<th>Up</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>o1</th>
<th>pic1.gif</th>
<th>o2</th>
<th>new.gif</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jim Hatch</td>
<td>0.77</td>
<td>0.83</td>
<td>Ted Jones</td>
</tr>
<tr>
<td>Dave Fox</td>
<td>0.17</td>
<td>0.23</td>
<td>Ken Yip</td>
</tr>
</tbody>
</table>
Generalized R-trees

- Previous representation does not provide an efficient way to perform nearest neighbor searches.
- Why? Only 2-attributes (bounding rectangles) are stored.
- In general, an object $o$ has an associated $(n+2)$-dimensional vector.
- If each point in the 2-d rectangle has $n$ attributes, then we need to use a generalized rectangle.
- A generalized rectangle for a space of dimensionality $g$ (specifically consider $g = n + 2$) may be defined by a set of constraints of the form:

$$
\ell_1 \leq x_1 \leq u_1 \\
\ell_2 \leq x_2 \leq u_2 \\
\vdots \\
\ell_g \leq x_g \leq u_g
$$

- When $g = 2$, we have $n = 0$, and in this case, an ordinary 2-dimensional rectangle is a special case of this definition.
- Sets of generalized rectangles are represented by generalized $R$-trees, or gR-trees.
gR-Trees

A generalized R-tree (gR-tree) of order \( K \) is exactly like an R-tree except for the following factors:

- First, each node \( N \) represents a generalized bounding rectangle \( GBR(N) \) of dimensionality \( (n + 2) \), which is represented by \( 2 \times (n + 2) \) real number fields, one for the lower bound and upper bound, respectively, of each dimension.

- When a node \( N \) is split, the union of the generalized bounding rectangles associated with its children equals the generalized bounding rectangle associated with \( N \).

- Each node (other than the root and the leaves) contains at most \( K \) generalized bounding rectangles and at least \( \lceil K/2 \rceil \) generalized rectangles.

- As usual, all \( (n + 2) \)-dimensional "data" rectangles are stored in leaves.
Nearest Rectangular Neighbors

• Suppose $R_Q$ is a query rectangle (which may represent an image object).

• Find all rectangles in a gR-tree $T$ that are as close to $R_Q$ as possible (where closeness is defined by a metric $d$ on points).

• We extend the metric $d$ to apply to rectangles as follows:

$$d(R, R') = \min\{d(p, p') \mid p \in R, p' \in R'\}.$$
Algorithm

**Algorithm 1** $\text{NN\_Search\_GR}(T, R_Q)$

$SOL = \text{NIL}; \ (\ast \text{ no solution so far } \ast);$  
$Todo = \text{List containing } T \text{ only};$  
$Bestdist = \infty; \ (\ast \text{ distance of best solution from } R_Q \ast);$  
while $Todo \neq \text{NIL}$ do 

{ 
  $F = \text{first element of } Todo;$  
  $Todo = \text{delete } F \text{ from } Todo;$  
  if $d(\text{GBR}(F), R_Q)) < Bestdist$ then 

  { 
    Compute children $N_1, \ldots, N_r$ of $F;$  
    if $N_i$’s are leaves of $T$ then 

      { 
        $N_{\text{min}} = \text{any } N_i \text{ at minimal distance from } R_Q;$  
        $\text{Ndist} = d(\text{GBR}(N_i), R_Q));$  
        if $\text{Ndist} < Bestdist$ then 

          { 
            $Bestdist = \text{Ndist}; \ SOL = N_{\text{min}};$  
          }  
        }  
    else $Todo = \text{insert all } N_i$’s into $Todo$ in order of distance from $R_Q;$ 

  }  
}

Return $SOL;$
end
Retrieving Images by Spatial Layout

Given an image $I$, and two objects (represented by rectangles) $o_1$, $o_2$ in $I$, a user may wish to ask queries of the form:

1. Is $o_1$ to the South of $o_2$?
2. Is $o_1$ to the South-East of $o_2$?
3. Are $o_1$ and $o_2$ overlapping?
### One-dimensional Precedence Relations

<table>
<thead>
<tr>
<th>o1</th>
<th>o2</th>
<th>Description</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>o1</td>
<td>o2</td>
<td>o1 before o2</td>
<td>B(o1,o2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o1 meets o2</td>
<td>M(o1,o2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o1 overlaps o2</td>
<td>OV(o1,o2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o1 during o2</td>
<td>D(o1,o2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o1 starts o2</td>
<td>S(o1,o2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o1 finishes o2</td>
<td>F(o1,o2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o1 equal o2</td>
<td>EQ(O1,o2)</td>
</tr>
</tbody>
</table>
2-Dimensional Precedence Relations

It is easy to extend the precedence relation along one dimension, to the case of two dimensions, is straightforward. If we use the notation $o[x]$ and $o[y]$ to denote the projection of object $o$ on the $x$ and $y$ axes, respectively, then it is easy to capture our spatial relationships as follows:

1. We say $o_1$ is **South** of $o_2$ iff $B(o_1[y], o_2[y])$ and either (i) $D(o_1[x], o_2[x])$ or (ii) $D(o_2[x], o_1[x])$ or (iii) $S(o_1[x], o_2[x])$ or (iv) $S(o_2[x], o_1[x])$ holds or (v) $F(o_1[x], o_2[x])$ or (vi) $F(o_2[x], o_1[x])$ holds or (vii) $E(o_1[x], o_2[x])$ holds.

2. Likewise, we say that $o_1$ is to the **Left** of $o_2$ iff either (i) $B(o_1[x], o_2[x])$ holds or (ii) $M(o_1[x], o_2[x])$ holds.