Chapter 4

Integrating QSR into Syntactic Pattern Recognition

4.1 Qualitative Representation of Motion Patterns

4.1.1 Definition of Binary Motion Pattern

A binary motion pattern is a pattern that is defined between two objects. A movement pattern is characterised by certain parameters. Thus, parameters are characteristics that are used in the definition of a motion pattern.The term *parameter*, in Pattern Recognition, refers to the constants in a hypothesis function. The usage of the term *parameter* in this thesis is based on motion pattern definitions in the domain of GIScience. Typically, parameters like direction, speed, distance, spatial location etc. are used to characterise a movement pattern. Parameters depend on the application domain. These parameters are dynamic in the sense that the values of the parameters change over time. In the framework proposed in this thesis, each motion pattern parameter is represented using a *JEPD* set of binary qualitative relations. A motion pattern is modeled as a sequence of temporal states. Each temporal state is characterised by a set of values for the binary qualitative relations that are used to represent movement parameters of the pattern. For example, let us consider a motion pattern where

direction and distance are used as movement parameters. Let direction be modeled qualitatively using the qualitative direction relations defined in section 3.1. The set of qualitative distance relations is taken as { veryclose, close, near, far }. Then a temporal state may be *(Same+, Close)*. This temporal state will last in time for a certain duration and a state transition will occur when there is a change in any or both of the qualitative relations that characterise the state. For example, this state may change to *(Same, near)* when direction relations changes from *Same+* to *Same* and distance relation changes from *close* to *near*.

Definition 4.1.1 *A temporal state of a motion pattern is an ordered n-tuple* $< r_1 , r_2 , . . . , r_n >$ *where each* r_i *is the value of a binary qualitative relation* R_i *used to represent the* i-th *movement parameter of the pattern. Each temporal state holds for a finite duration of time.*

Temporal states are the building blocks using which a motion pattern can be represented. Temporal states are assigned a qualitative interpretation by defining them in terms of qualitative relations. A transition from one state to another is caused by a change in any relation value r_i in the state. In our framework, each temporal state is treated as symbol of some alphabet set Σ of a language. In formal language theory, a language is defined as a set of strings. In state-space representation, a motion pattern can be represented as a sequence of temporal states. When we move from the state-space representation to the language-theoretic representation, it is necessary to treat each temporal state as a symbol of some alphabet Σ of a language. Then, a sequence of temporal states becomes a sequence of symbols i.e. a string of symbols. Symbols that comprise such a string are terminals of a language. This is the language that represents the motion pattern in language-theoretic domain. Different instances of the pattern are represented by different strings. By instances, we refer to the different ways in which a motion pattern can occur. All these strings, representing various instances of the pattern, belong to the same language that describe the pattern. A formal grammar can be used to define such a language. In subsequent sections, we further explore this language-theoretic interpretation of a motion pattern.

Definition 4.1.2 *Binary Motion Pattern: A binary motion pattern* ξ *for two*

objects can be denoted by (G_{bmp}, χ), *where* G_{bmp} *is a grammar and* χ *is a set where each element is a Jointly Exhaustive Pairwise Disjoint (*JEPD*) set of binary qualitative spatial relations.*

Definition 4.1.3 *The set* $\chi = \{ R_1 , R_2 , . . . , R_n \}$, *where each* R_i *is a* JEPD *set of binary qualitative spatial relations. Each* R_i *represents the* i-th *parameter of the movement pattern qualitatively.*

A binary motion pattern holds between two objects. There are two important considerations in the definitions given above. One is about how each movement parameter is represented. Each movement parameter is represented by a *JEPD* set of binary qualitative spatial relations. The other is a grammar termed as G_{bmp}. The properties of G_{bmp} will be gradually explored in subsequent discussions. A binary motion pattern can also be defined as a formal language:

Definition 4.1.4 *A binary motion pattern between two objects can be defined as a set* $L_{bmp} = \{ w \mid G_{bmp} \Longrightarrow^* w \}$

We would like to discuss now how a state-space representation of a motion pattern can be transformed into a string of symbols. For this purpose, we introduce a notion called primitive patterns:

Definition 4.1.5 *Primitive Pattern : Let* $R_1 , R_2 , . . . , R_n$ *be* JEPD *sets of binary qualitative spatial relations, each* R_i *representing a different movement parameter qualitatively. Then, a primitive pattern between a primary object and a reference object is denoted by an n-tuple of the form* $\prec r_1 , r_2 , ... r_n \succ$ *such that* $\prec r_1 , r_2 , ... r_n \succ \in R_1 \times R_2 \times . . . R_n$.

Each primitive pattern is a temporal state that can be called as a snapshot of the motion pattern during a certain time interval. Primitive patterns form the the alphabet set Σ_{bmp} of the grammar G_{bmp}. Each primitive pattern is a terminal symbol using which strings in the language L_{bmp} are constructed. It is obvious that the set of primitive patterns is exhaustive i.e. it includes all possible temporal states a motion pattern may be in.

4.1.2 Spatio-temporal Continuity

Spatio-temporal continuity, expressed in the form of conceptual neighbourhood graphs, imposes an ordering on the terminals in the set Σ_{bmp}.

Definition 4.1.6 *Neighbouring Terminal: Let Σ_{bmp} be the alphabet set for the grammar G_{bmp}.Let $a \in \Sigma_{bmp}$ and let a be of the form $\prec r_1$, r_2 , ... $r_n \succ$. Let $b \in \Sigma_{bmp}$ and it is of the form $\prec s_1$, s_2 , ... $s_n \succ$. Then, b is a neighbouring terminal of a iff s_1 is a conceptual neighbour of r_1, s_2 is a conceptual neighbour of r_2, ... and s_n is a conceptual neighbour of r_n.*

Therefore, spatio-temporal continuity restricts the terminals that may appear after a given terminal in a string representation of a motion pattern. If a and b are consecutive terminals in a string, then it must be the case that b is neighbouring terminal of a and vice-versa. This notion of continuity influences the productions of G_{bmp} i.e. it affects the form of grammar necessary to recognize a binary motion pattern. Moreover, spatio-temporal continuity of terminal symbols can be exploited to handle low level processing errors and missing observations.

Spatio-temporal continuity affects the format of a string in the language L_{bmp}. The following lemma states an important property about a string $w \in L_{bmp}$.

Lemma 4.1.1 *Let L_{bmp} be a binary motion pattern language defined over Σ_{bmp} and let* a $\in \Sigma_{bmp}$. *Let* w *be a string representation of a motion pattern and let* w $\in \Sigma^*$. *Then,* w *can not have a substring of the form a^n where* n *is greater than or equal to two.*
Proof: The proof follows from the fact that in QSR*, change is recorded only when it crosses a qualitative boundary. Let the terminal* a *be of the form $\prec r_1$, r_2 , ... $r_n \succ$. When any qualitative relation r_i changes to its conceptual neighbour, then only this change is recorded and a new terminal results. So, it is not possible that the same terminal* a *occurs more than once consecutively in a string.*

The above lemma simplifies algorithms for representation of a motion pattern string using productions of a grammar. The lemma states that the same

terminal is not repeated multiple times consecutively in string representation of a motion pattern. After a terminal, its neighbouring terminal appears in the string. Because of this, the representation of the string using productions of a regular grammar becomes very simple and consequently, the algorithm for learning these grammar productions becomes simple. This is so because when the above lemma holds, we can treat all terminals in the same manner and add a production like $A \rightarrow a\ B$ where A and B are non-terminals and a is the terminal under consideration. With repetition of a multiple times consecutively, we need to add appropriate productions to generate the repetition. So, for every terminal encountered in the string, we need to check whether it is a single occurrence or a repetition.

4.2 Integration of QSR with Formal Grammar

4.2.1 Definition of G_{bmp}

Definition 4.2.1 *The grammar $G_{bmp} = (\ V, \Sigma_{bmp}, P, S\ )$, where V is a set of non-terminal symbols, Σ_{bmp} is a set of terminal symbols, P is the set of productions and S is the start symbol of the grammar.*

Spatio-temporal continuity of terminal symbols in Σ_{bmp} makes it possible to use a regular grammar for representation and recognition of L_{bmp}.

Lemma 4.2.1 *The motion pattern grammar $G_{bmp} = (\ V, \Sigma_{bmp}, P, S\ )$ is a regular grammar i.e. each production is of the form $A \rightarrow a\ B$ where $a \in \Sigma_{bmp}$ and $A, B \in V$.*

Proof: Let m be a motion pattern and let m be of the form $a_1\ a_2\ a_3\ .\ .\ .\ a_n$, where each $a_i \in \Sigma_{bmp}$. Each a_{i+1} is a neighbouring terminal of a_i. For recognising m, we can construct a deterministic finite automaton with states S_0, $S_1\ .\ .\ .\ S_n$ such that there is a transition from state S_i to state S_{i+1} on input a_{i+1}. Here, S_0 is the start state and S_n is the final state. From equivalence of DFA and regular grammar, we can write grammar productions as $S_0 \rightarrow a_1\ S_1$, $S_1 \rightarrow a_2\ S_2$, . . . $S_{n-1} \rightarrow a_n\ S_n$, $S_n \rightarrow \epsilon$. Here, each S_i is a non terminal.

The above lemma is significant in recognition of a binary motion pattern. As the motion pattern grammar G_{bmp} is a regular grammar, this grammar can be converted into a finite state automaton during recognition and therefore, recognition of a binary motion pattern can be done in linear time using this finite state automaton.

4.2.2 Implication of JEPDness

The set of binary qualitative relations used for representing a movement parameter should be Jointly Exhaustive and Pairwise Disjoint (*JEPD*). Jointly Exhaustiveness (JE) property affects the alphabet set Σ_{bmp} of the motion pattern language L_{bmp}. For example, let us assume that L is a language and T be its alphabet set. Let G_L be the grammar that recognises L. Let $T = \{$ a, b, c $\}$. Therefore, all the strings that belong to the language L will be formed out of the elements of the set T. If some string w contains a symbol $d \notin T$, then L can not be recognised by G_L. This implies that the alphabet set must be exhaustive in the sense that it must contain all the symbols that may constitute strings of the language L. For the binary motion pattern language L_{mp}, any terminal symbol $a \in R_1 \times R_2 \times \ldots R_n$. Therefore, for the alphabet set Σ_{bmp} to be exhaustive, each R_i must be exhaustive. This is stated in the following lemma:

Lemma 4.2.2 *The motion pattern grammar $G_{bmp} = (\ V\ ,\ \Sigma_{bmp}\ ,\ P\ ,\ S\ )$ can recognize the language L_{bmp}, only when each binary qualitative spatial relation R_i representing the* i-th *movement parameter is Jointly Exhaustive.*
Proof: Let us assume that some R_l is not Jointly Exhaustive. This means there may be some binary qualitative relation r_l that can act as the value of the $l-th$ movement parameter, but $r_l \notin R_l$. So, the terminal $b = \prec r_1\ ,\ r_2\ ,\ ...r_l,\ r_{l+1}\ \ldots\ r_n \succ \notin \Sigma_{bmp}$. Let $w \in L_{bmp}$ and w be of the form $a_1,\ a_2,\ \ldots\ b\ \ldots\ a_n$. For this w, there can not be a production of the form $X \rightarrow b\ Y$, where $X,\ Y \in P$. So, w can not be parsed by G_{bmp} and hence L_{bmp} can not be recognised by G_{bmp}.

For recognition of a string by a grammar, input is scanned over one symbol at a time. The symbol currently under consideration decides which production rule

to use for parsing. If the current input symbol is not known deterministically i.e. if it is possible that current input is not a single terminal, but a set of terminals, then there is non determinism in parsing. Since G_{bmp} parses deterministically, it is necessary that at any point of time a single terminal should be under consideration. For this, the Pairwise Disjointedness (PD) of binary qualitative relations is an important property.

Lemma 4.2.3 *The grammar G_{bmp} parses deterministically when each binary qualitative spatial relation R_i representing the* i-th *parameter of the motion pattern is Pairwise Disjoint.*
Proof: Let us assume that some R_l is not Pairwise Disjoint. This means that there can be two relations r_{ql}, $s_{ql} \in R_l$ such that both can hold at the same point of time. Let a, $b \in \Sigma_{bmp}$. Let $a = \prec r_1 , r_2 , ...r_{ql} \ . \ . \ . \ r_n \succ$ and $b = \prec r_1 , r_2 , ...s_{ql} \ . \ . \ . \ r_n \succ$. Since, r_{ql} and s_{ql} can hold at the same point of time, it is possible that the terminals a and b occur simultaneously. Therefore, the production for parsing the current input needs to be non deterministically chosen.

The above lemmas are significant because they state an important property of the binary qualitative relations that can be used for each movement parameter. If the set of binary qualitative relations, used for a movement parameter, is not *JEPD*, then deterministic parsing of a binary motion pattern using grammar productions is not possible.

4.2.3 Interpretation of Motion Patterns in Temporal Domain

A motion pattern lasts for a finite interval of time. Since a motion pattern consists of one or more primitive patterns, the time duration of the whole pattern depends on the individual time taken by each of its constituent primitive pattern. If the time taken by each constituent primitive pattern is known, then it is possible to know the time for whole motion pattern. The time taken by a motion pattern has been termed as its *Hold Interval*.

Definition 4.2.2 *Hold Interval: Let ψ be a binary motion pattern and ψ be of the form a_1 , a_2 , . . ., a_n, where each $a_i \in \Sigma_{bmp}$. Let T_i be the time interval for which the primitive pattern a_i holds. Let t_1 be the start time for the interval T_1 and t_n be the end time for the interval T_n. Then, the interval whose start time is t_1 and end time is t_n is called the Hold Interval for the binary motion pattern ψ*

The time taken by each primitive pattern can be computed during input processing. The *HoldInterval* for the entire motion pattern can be computed during parsing using a syntax directed translation scheme.

4.3 Automatic Learning of a Binary Motion Pattern

4.3.1 Training Data Format

In order to have a formal representation of a motion pattern in the form of a grammar, the framework proposed in this thesis provides two methods. The first is to describe the pattern using a program written in a qualitative language. This approach is explained in chapter 5. The other is to learn the definition of the pattern in the form of a formal grammar from a set of training examples. For learning a binary motion pattern, it is necessary to learn the production rules automatically from training data set. Spatio-temporal relations of the primary object with respect to the reference are computed from the training data set. The training data set may be real life or synthetic. As an example of real life data set, we can cite the case of collecting data about a particular motion pattern from video samples and recording these data in a pre-defined format for learning. Similar synthetic data set can be generated through simulation. For manually designed training data, we need to have an idea about the different ways in which the motion pattern may manifest. Whatever may be the method of data collection, the commonality is that the specifications of the pattern from different training data sets are recorded in a pre-defined format. This pre-defined format

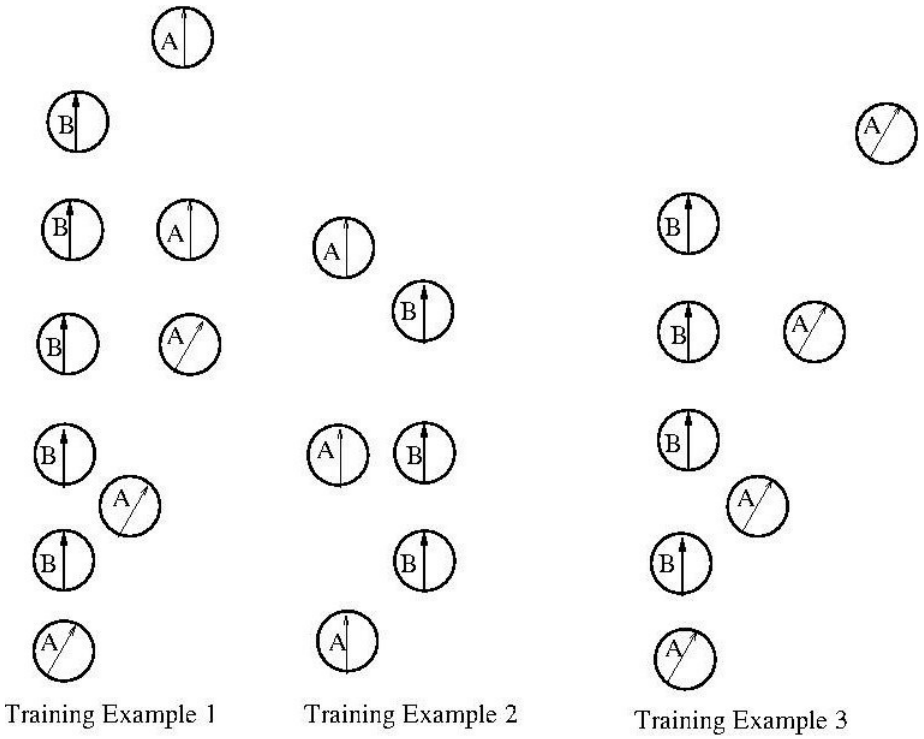

Figure 4.1: Training Examples for Overtake Pattern

for a single training record is like:

```
Pattern_Name=
No_Of_Objects=
Object_List=
Reference=
Object_Id=Pattern_Specification;HoldInterval_Start;HoldInterval_End
Object_Id=Pattern_Specification;HoldInterval_Start;HoldInterval_End
...
```

The first line of this specification is common to all training data. This line tells the name of the binary motion pattern for which the training data are provided. The second line in each record gives the number of objects in that particular example. The third line specifies the identity of each object participating in the pattern and the fourth one specifies the reference. After this, there is a specification of a binary motion pattern along with its *HoldInterval* for each object with respect to the reference.

An example of such training data for an *overtake* pattern is presented below. We assume that there are two objects A and B and the object B is the reference. Objects are abstracted as directed rectangles. Spatial orientation and direction are taken as movement parameters. These two are represented using the *JEPD* sets explained in chapter 3. In all the three training examples given below, the object A is initially behind the object B. In the first example, the object A goes to the right of B from behind and finally from right it goes to the front of B. In the second training example, from behind, A goes to the left of B and from left it finally goes to front. In the third case, A is beside B on left. From there, it goes to front. In the training examples, these cognitive descriptions of the three overtake examples have been translated into string representation using primitive patterns. Since time is represented only at the level of a pattern (not at the level of each primitive pattern i.e. a state), the hold interval for the entire pattern is indicated at the end of the binary motion pattern by listing the end points of this interval.

```
Pattern_Name=Overtake
No_Of_Objects=2
Object_List=A;B
Reference=B
A=(Back,Same-),(BackRight,Same-),(Right,Same-),(Right,Same),
(FrontRight,Same);10;30

Pattern_Name=Overtake
No_Of_Objects=2
Object_List=A;B
Reference=B
A=(BackLeft,Same),(Left,Same),(FrontLeft,Same);20;50

Pattern_Name=Overtake
No_Of_Objects=2
Object_List=A;B
```

```
Reference=B
A=(Back,Same-),(BackRight,Same-),(Right,Same-),
(FrontRight,Same-);15;30
```

Three training records have been manually designed. This specification is based on our intuitive idea of an overtake. In Figure 4.1, these overtake patterns have been illustrated.

4.3.2 Algorithm For Learning

It is possible to learn the production rules of grammar G_{bmp} automatically from a training data set. A learning algorithm is presented below. This algorithm takes a training data set in pre-defined format and constructs the productions of the grammar G_{bmp}. The Pattern_Specification field in a training record is assumed to be of the form $a_1 a_2 \ldots a_n$.

Algorithm 1 Learn_bmp(Training_Data)

1: **while** there is a training record **do**
2: j:=1
3: Read training record
4: **for** each object A_i in Object_List **do**
5: **if** there is no production of the form $S \rightarrow A_i$ **then**
6: add $S \rightarrow A_i$
7: add $A_i \rightarrow X_j$
8: add $X_j \rightarrow a_1\ X_{j1}$, $X_{j1} \rightarrow a_2\ X_{j2}$, . . . $X_{j(n-1)} \rightarrow a_n\ X_{jn}$, $X_{jn} \rightarrow \epsilon$
9: j:=j+1
10: **else**
11: Search Pattern_Specification string in production list of A_i
12: **if** a match is not found **then**
13: add the production $A_i \rightarrow X_j$
14: add the production $X_j \rightarrow a_1\ X_{j1}$, $X_{j1} \rightarrow a_2\ X_{j2}$, . . . $X_{j(n-1)} \rightarrow a_n\ X_{jn}$, $X_{jn} \rightarrow \epsilon$
15: j:=j+1
16: **end if**
17: **end if**
18: Read HoldInterval of the binary motion pattern for object A_i
19: **end for**
20: Find_Max_HoldInterval_bmp()
21: **end while**

In this algorithm, a non-terminal is used for each object. This non-terminal has the same name as the name of the object. When an object is seen in a training record for the first time, a production is added from the start symbol to the non-terminal representing that object. The format of a string in Pattern_Specification field is assumed as $a_1 a_2 \ldots a_n$. In line 8 of the algorithm, productions are added for parsing a pattern string corresponding to the object under consideration when the object is seen for the first time in a training record.

Otherwise if an encountered object was seen before, it is necessary to check whether the binary motion pattern stored in *Pattern_Specification* string is already learned from some earlier training example for that object. For this, we search for *Pattern_Specification* string in the set of productions for that object. We take this view that *Pattern_Specification* string match terminal by terminal starting at the first position. If such a match is found, it means that the current example (stored in *Pattern_Specification* string) was already seen exactly in the same way or it occurred within an earlier example. It is true that even when the current example in Pattern_Specification string matches with earlier example(s), the temporal characteristics may be quite different state wise. In the framework proposed in this thesis, time is not stored at the level of a terminal (state). Therefore, when a match is found, we assume that this example is already seen and accordingly, do not add any production for this. Otherwise, if no match is found, then this example has to be stored as a different way of performing the event and this is done in line number *13* to *15* of the algorithm. The maximum interval of time for which the binary motion pattern occurs is calculated by the function *Find_Max_HoldInterval_bmp()*. Since hold intervals of all the **bmp**s are available in training data set, the problem is to find the maximum among these individual **bmp**s.

Illustrative Example

As an example of operation of this algorithm, we look at the training data set given for the *overtake* pattern in 4.3.1. When the first record is encountered, the following productions are added:

$S \rightarrow A_1 \mid A_2 \mid A_3$

$A_1 \rightarrow X_1$

$X_1 \rightarrow$ (Back,Same-) X_{11}

$X_{11} \rightarrow$ (BackRight,Same-) X_{12}

$X_{12} \rightarrow$ (Right,Same-) X_{13}

$X_{13} \rightarrow$ (Right,Same) X_{14}

$X_{14} \rightarrow$ (FrontRight,Same) X_{15}

$X_{15} \rightarrow \epsilon$

The second record specifies that possibilities exist at the first two terminals. The productions are augmented as:

$A_2 \rightarrow$ (BackLeft,Same) X_2

$X_2 \rightarrow$ (Left,Same) X_{21}

$X_{21} \rightarrow$ (FrontLeft,Same) X_{22}

$X_{22} \rightarrow \epsilon$

The third training record shows a new possibility for the third terminal. The production set is augmented as:

$A_3 \rightarrow$ (Back,Same-) X_3

$X_3 \rightarrow$ (BackRight,Same-) X_{31}

$X_{31} \rightarrow$ (Right,Same-) X_{32}

$X_{32} \rightarrow$ (FrontRight,Same-) X_{33}

$X_{33} \rightarrow \epsilon$

Complexity of Learning

When an object is seen for the first time, we need to store productions for each terminal in the Pattern_Specification string.

Let L_P be the maximum length of the Pattern_Specification string. When the object is seen for the first time, L_P number of productions are stored. Storing of a production can be done in constant time. In the second case where the object was already seen, we need to check whether the Pattern_Specification string

was already seen as a prefix of some already learned example. The number of examples learned already can not be more than the total number of training examples. Let the total number of training examples be N_T. So, this search takes time O($L_P * L_P * N_T$). Let N_O be the number of objects in a training record. Therefore, searches for all the examples in the training data set can not take time more than O($L_P * L_P * N_T * N_T * N_O$). The number of objects does not vary with training examples and it is same for all training examples. Therefore, this time complexity can be approximated as O($L_P * L_P * N_T * N_T$).

4.4 Handling Low Level Error in Learning

4.4.1 The Principle

There are two types of errors that may be encountered in low level processing. Firstly, we may encounter a primitive that is not a conceptual neighbour of its previous primitive pattern. This may happen at the time of training the framework and also at the time of recognition of a learned motion pattern. The second is the problem of missing observations. In video processing, problem can be caused by obstruction of camera view by other objects in such a way that one or more objects may be occluded. The details about movements of objects during the occluded period are not known definitely.The framework proposed in this thesis is based on the notion of spatio-temporal continuity of binary qualitative relations. It is expected that after a given primitive, its conceptual neighbour will be encountered next. Because of above mentioned errors in low level processing, this may not always hold.

In our framework, we treat both these types of errors as instances of discontinuity. If we encounter a primitive q after p, but q is not a neighbouring primitive of p, then there is a discontinuity in input. It is not important whether this happened because of processing error (for example in object detection, tracking etc.) or occlusion. Using spatio-temporal continuity of binary qualitative relations, we try to infer what may be a possible sequence of primitives between p and q. There is no definite way of knowing the actual sequence because the

same discontinuity may be caused by different types of errors. If the technique of handling the discontinuity remains same during learning and recognition, then all kinds of errors that caused the discontinuity are automatically included in the definition of the pattern. We follow this principle in our framework.

We try to find a sequence of primitives to fill the discontinuity between p and q. The problem is to find a string $a_1a_2 \ldots a_n$ such that $pa_1a_2 \ldots a_nq$ satisfies spatio-temporal continuity. This means that a_1 is a neighbouring primitive of p, any a_{i+1} is a neighbouring primitive of a_i for i in the range from *1* to $n-1$ and q is a neighbouring primitive of a_n. Whenever there is a discontinuity between p and q, the same string $a_1a_2 \ldots a_n$ is used to achieve continuity. By doing this, we always treat the error in the same manner and incorporate the error into the representation of the learned pattern. During recognition, when an error in the form of two consecutive primitives p and q is observed, we know that this error was seen during learning and steps were incorporated into the productions for handling this kind of discontinuity. In this approach, we need to be careful about the fact that the string $pa_1a_2 \ldots a_nq$ may be observed during recognition though it is not a learned string. Actually, the discontinuous string with p followed by q is the learned string; the string $a_1a_2 \ldots a_n$ is artificially filled during learning. So, there is a possibility that false patterns may be recognised.

Extra information needs to be incorporated in the primitives so that the exact path $a_1a_2 \ldots a_n$ can be traced. This extra information is explicitly maintained in the learned pattern only. This extra information contains two fields. The first is called *category*. The category specifies whether the primitive was a conceptual neighbour of its previous primitive or not. If yes, then the value of the category field is N to indicate that it is a non-error case. Otherwise, the category field is set to the value E to indicate that it is an error case. When the primitive is of category E, then we store the starting and ending primitives for the discontinuity. We name this information as $Path\_Name$. Otherwise, if the primitive category is N, then $Path\_Name$ is set to a *Null* value to indicate its non-significance. As an example, let us consider the discontinuity between p and q. When this is filled with the sequence $a_1a_2 \ldots a_n$, each a_i is of category

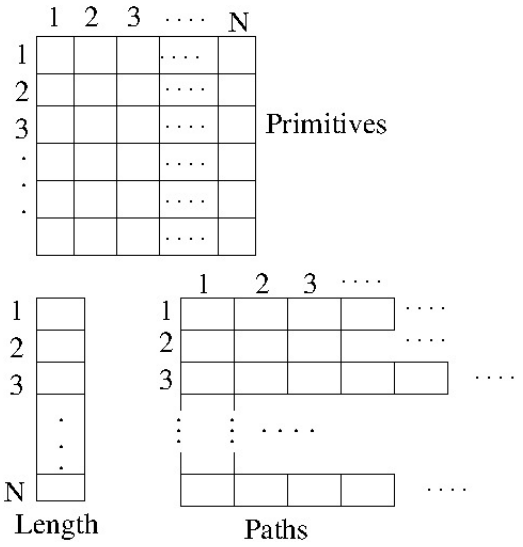

Figure 4.2: Data Structures for Error Handling During Learning

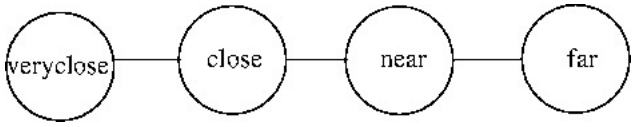

Figure 4.3: Conceptual Dependency of Distance Relations

E. The $Path\_Name$ information in each of these a_i is $< p, q >$. These two fields i.e. *Category* and $Path\_Name$ are auxiliary information maintained with the primitives in order to prevent recognition of false patterns.

4.4.2 Algorithm for Handling Discontinuity

We present an algorithm below for finding a sequence of primitive patterns whenever a discontinuity is observed in Pattern_Specification field during learning. Data structures used in this algorithm are shown in Figure 4.2. In the table *Primitives* in the figure, the primitive patterns computed during the operation of the algorithm are stored. If there are N binary qualitative relations representing movement parameters, then each primitive will have N relation values. Therefore, the number of columns in the *Primitives* table is N. Each row of the *Paths* table is used for handling discontinuity at a particular position in the primitive. For example, Paths[1] gives us the sequence of relation values for the first binary qualitative relation R_1.

Algorithm 2 Find_Sequence(p,q,Len)

Two primitive patterns p and q are passed as arguments. After p, q is encountered and q is not a neighbouring terminal of p. The primitive p is of the form (p_1, p_2, . . . p_n) and the primitive q is of the form (q_1, q_2, . . . q_n). The binary qualitative relations for the movement parameters are R_1,R_2, . . . R_n. The algorithm stores the computed sequence of primitives in a data structure called *Primitives*. The length of the sequence is returned in the parameter named as *Len*.

```
1: for i:= 1 To n do
2:     Paths[i] := Find_Shortest(p_i,q_i)
3:     Length[i] := Find_Length(Paths[i])
4: end for
5: Max_Length := Length[1]
6: for i := 2 To n do
7:     if Length[i] > Max_Length then
8:         Max_Length := Length[i]
9:     end if
10: end for
11: for j := 1 To n do
12:     for i := 1 To Length[i] do
13:         Primitives[i][j]:= Paths[j][i]
14:     end for
15:     if Length[j] < Max_Length then
16:         for k := Length[j]+1 To Max_Length do
17:             Primitives[k][j]:= Paths[Length[j]]
18:         end for
19:     end if
20: end for
21: Len := Max_Length
```

In line 2 of the algorithm, we compute the shortest paths between corre-

sponding relation values in the primitive patterns using conceptual neighbourhood graphs. For example, shortest paths between p_1 and q_1 is computed using the conceptual neighbourhood graph of R_1. Each such path is in the form of sequence of primitives. In line 3 of the algorithm, any such path, computed between p_i and q_i is stored in Paths[i]. Maximum length of such a path is computed and stored in Max_Length in lines 5 to 10 of the algorithm. From the *Paths* data structure, these shortest paths are assigned to the *Primitives* data structure in lines 11 to 20 of the algorithm. For the paths that have length less than the maximum path length observed between two primitives (available in Max_Length), the last relation value is repeated. This is done in lines 15 to 19.

In the work of Fernyhough [133], handling of *noise* using a continuity network (conceptual neighbourhood) is reported. It is assumed and motion is continuous and the continuity network is respected. In their approach, in the event history verification step, history is *fixed* by inserting a missing relationship tuple or by removing an extraneous one. However, when the discrepancy is large, the entire sequence is discarded. There are similarities and differences with our approach of handling *noise* using conceptual neighbourhood. We too assume that motion is continuous and if a pattern occurs, it must occur in a way that respects continuity networks. Accordingly, missing primitives are inserted between *discrepancies*. In our approach, we do not remove any discrepancy. Moreover, we do not discard the sequence. If the *discrepancies* are same, then we the same sequence of missing primitives is inserted always. This is like trying to represent the *noise* also in the pattern in the hope that if the same *noise* occurs during recognition, the same treatment is applied and resulting sequences too become identical.

Complexity of handling discontinuity

When discontinuity is observed, the algorithm needs to compute shortest path from each p_i to each q_i in the conceptual neighbourhood graph of the corresponding *JEPD* set of binary qualitative relations. Shortest paths between all-pairs of nodes in a conceptual neighbourhood graph can be pre-computed and stored in a hash table along with its length. In that case, finding the required shortest

path between any p_i and p_i takes constant time and the loop from line 1 to 4 takes $O(n)$ time. Here, n is the number of movement parameters. In lines from 5 to 10, the maximum among these path lengths is computed and this is done in $O(n)$ time. If this maximum path length is denoted by L_M, then the nested loop from lines 12 to 14 runs in $O(L_M)$ time. In the worst case, it can be assumed that the loop in line 15 too tuns for $O(L_M)$ time. Therefore, complexity of the for loop in line 11 can be taken as $O(n * L_M)$ in the worst case. So, the complexity of algorithm 2 in the worst case is $O(n * L_M)$. Since the maximum path length between any two nodes for all conceptual neighbourhood graph can be computed in advance, the parameter L_M can be considered to be a constant integer. Moreover, for a particular application the number of movement parameters is also fixed. Therefore, for a particular application discontinuity can be handled in constant time.

Illustrative Example

For an example of operation of this algorithm, we choose qualitative direction and distance as two movement parameters. Qualitative directions are modeled using the *JEPD* set of relations introduced in section 3.1. For convenience, we refer to this set as Q. The set $D = \{$ *very close*, *close*, *near*, *far* $\}$ is used to model distance qualitatively. The conceptual neighbourhood graph for these distance relations is shown in Figure 4.3. We take two primitive patterns p and q such that $p =$ (*Same* , *close*) and $q =$ (*opposite+* , *far*). Obviously, q is not a neighbouring pattern of p. In the conceptual neighbourhood graph in Figure 4.3, the shortest path between *close* and *far* is *close*, *near*, *far*. The length of this path is two. In the conceptual neighbourhood graph in Figure 3.5, we observe that the shortest path between *Same* and *opposite+* is *Same*, *Same-*, *lr+*, *lr*, *lr-*, *opposite+*. The length of this path is five. Therefore, the missing primitives between p and q are constructed as: (*Same-* , *near*), (*lr+* , *far*), (*lr* , *far*), (*lr-* , *far*), (*opposite+* , *far*). The *Category* of each of this generated primitives is E and the $Path\_Name$ field is set to (*Same* , *close*), (*opposite+* , *far*). During recognition, if the same discontinuity is observed between consecutive primitives

p and q, missing primitives will be generated using the same algorithm.

4.4.3 Modification of the Learning Algorithm

We need to incorporate error handling into the learning algorithm for a binary motion pattern. Whenever a new terminal is seen in the training input, it is necessary to check whether this terminal is a neighbouring terminal of its predecessor.

Algorithm 3 Learn_bmp_Modified(Training_Data)

The value of Pattern_Specification field in a training record is assumed to like a_1, a_2, . . . a_n.

```
1: while there is a training record do
2:     Read training record
3:     j := 2
4:     Sequence := ""
5:     for each object A_i in current training record do
6:         for each primitive a_j in Pattern_Specification do
7:             if a_j is not a neighbouring terminal of a_{j-1} then
8:                 Find_Sequence(a_{j-1},a_j,Len)
9:                 for k:=1 to Len do
10:                    Set Category of Primitives[k] to E
11:                    Set Path_Name of Primitives[k] to (p, q),
12:                    Concatenate(Sequence,Primitives)
13:                end for
14:                Insert Sequence into Pattern_Specification at index j
15:                j := j + Length +1
16:            else
17:                j := j +1
18:            end if
19:        end for
20:    end for
21: end while
```

22: **while** there is a training record **do**
23: j:=1
24: Read training record
25: **for** each object A_i in Object_List **do**
26: **if** there is no production of the form $S \rightarrow A_i$ **then**
27: add $S \rightarrow A_i$
28: add $A_i \rightarrow X_j$
29: add $X_j \rightarrow a_1\ X_{j1}$, $X_{j1} \rightarrow a_2\ X_{j2}$, . . . $X_{j(n-1)} \rightarrow a_n\ X_{jn}$, $X_{jn} \rightarrow \epsilon$
30: j:=j+1
31: **else**
32: Search Pattern_Specification string in production list of A_i
33: **if** a match is not found **then**
34: add the production $A_i \rightarrow X_j$
35: add the production $X_j \rightarrow a_1\ X_{j1}$, $X_{j1} \rightarrow a_2\ X_{j2}$, . . . $X_{j(n-1)} \rightarrow a_n\ X_{jn}$, $X_{jn} \rightarrow \epsilon$
36: j:=j+1
37: **end if**
38: **end if**
39: Read HoldInterval of the binary motion pattern for object A_i
40: **end for**
41: Find_Max_HoldInterval_bmp()
42: **end while**

In Algorithm 3, for each object we remove discontinuities present in Pattern_Specification field of a training record. We check whether a terminal a_j is a neighbouring terminal of a_{j-1}. If not, we compute the sequence of terminals by making a call to *Find_Sequence*. *Find_Sequence* returns the sequence of terminals in a data structure called *Primitives*. We concatenate these terminals into a variable *Sequence* and insert the resulting string into Pattern_Specification field at index *j*. The length of Pattern_Specification increases after this insertion and therefore, we update the index variable j to get to the next primitive to be

Basic Interval Relation	Sym-bol	Pictorial Example	Endpoint Relations
X before Y	$\prec$	xxx	$X^- < Y^-, \quad X^- < Y^+,$
Y after X	$\succ$	yyy	$X^+ < Y^-, \quad X^+ < Y^+$
X meets Y	m	xxxx	$X^- < Y^-, \quad X^- < Y^+,$
Y met-by X	$\mathsf{m}^\smile$	yyyy	$X^+ = Y^-, \quad X^+ < Y^+$
X overlaps Y	o	xxxx	$X^- < Y^-, \quad X^- < Y^+,$
Y overlapped-by X	$\mathsf{o}^\smile$	yyyy	$X^+ > Y^-, \quad X^+ < Y^+$
X during Y	d	xxx	$X^- > Y^-, \quad X^- < Y^+,$
Y includes X	$\mathsf{d}^\smile$	yyyyyyy	$X^+ > Y^-, \quad X^+ < Y^+$
X starts Y	s	xxx	$X^- = Y^-, \quad X^- < Y^+,$
Y started-by X	$\mathsf{s}^\smile$	yyyyyyy	$X^+ > Y^-, \quad X^+ < Y^+$
X finishes Y	f	xxx	$X^- > Y^-, \quad X^- < Y^+,$
Y finished-by X	$\mathsf{f}^\smile$	yyyyyyy	$X^+ > Y^-, \quad X^+ = Y^+$
X equals Y	$\equiv$	xxxx yyyy	$X^- = Y^-, \quad X^- < Y^+,$ $X^+ > Y^-, \quad X^+ = Y^+$

Figure 4.4: Allen's Interval Algebra: In terms of Endpoint Relations

considered. Once the discontinuities are eliminated, learning is identical to what was done in algorithm 1.

Complexity of Modified Learning Algorithm

Let us assume that the number of training records is N_T, the maximum number of objects in a any training record is N_O and the maximum length of any pattern specification string is L_P. Then the *for* loop in line *5* runs for $O(N_O)$ time, the *for* loop in line *6* runs for $O(L_P)$ time. The most frequent operations in this algorithm are steps in lines *10*, *11* and *12*. Out of these, steps in lines *10* and *11* can be carried out in constant time. The concatenation operation in line *12* joins all the computed primitives into a single string. If the maximum path length of a discontinuity is L_M, then the number of symbols to be copied is $O(L_M)$. Considering the fact that the *while* loop in line *1* runs for $O(N_T)$ time, the complexity of the most frequent operation at line *12* is $O(N_T * N_O * L_P * L_M)$. Since the conceptual neighbourhood graph for a set of *JEPD* relations is known a priori, the parameter L_M can be considered to be a constant. The part of this algorithm from line *22* to line *42* is same as what was done in algorithm 1. This part has the complexity O($L_P * L_P * N_T * N_T$). This is the dominant part of the algorithm and so, the complexity of algorithm 3 can be taken as O($L_P * L_P$

$* N_T * N_T$).

4.5 Basic Multi Object Pattern: Representation, Learning and Recognition

4.5.1 Definition and Representation

A binary motion pattern expresses a motion pattern between two objects. In reality, a motion pattern may involve more than two objects. A motion pattern among multiple objects will thus be naturally termed as *Multi Object Motion Pattern* (abbreviated as *MOP*). There is a subclass of *Multi Object Motion Pattern*s that we term as *Basic Multi Object Motion Pattern*. A *Basic Multi Object Motion Pattern* (abbreviated as *BMOP*)too has more than two participating objects. The difference is that a *BMOP* can be learned from training data because it does not contain any other nested *BMOP* within it. A *BMOP* is directly learnable from training examples because inside a *BMOP*, we allow binary motion patterns only and techniques for learning these from training data examples have been presented in algorithm 3.

Definition 4.5.1 *A basic multi object motion pattern (to be abbreviated as* bmop*)* Υ *is defined as a triple (M, O, T), where M is a set of binary motion patterns, O is a set of objects and T is a set of temporal constraints.*

The set M can be written as $M = (\ \xi_1, \xi_2, \ldots, \xi_n\ )$, where each ξ_i is a binary motion pattern between an object in the set O and a reference object in the same set O. Since all objects share the same set of qualitative movement parameters, the set χ i.e. the set of *JEPD* binary qualitative relations is same for all the binary motion patterns. When interpreted as a language, a basic multi object pattern will be a set of sets. Each set will contain the strings that are generated by some Gi_{bmp} for the binary motion pattern ξ_i.

Definition 4.5.2 *A basic multi object pattern can be defined as a language* $L_{bmop} = \{ \{ w1_{\xi_1}, w2_{\xi_1}, \ldots, wn_{\xi_1} \}, \{ w1_{\xi_2}, w2_{\xi_2}, \ldots wk_{\xi_2} \}, \ldots \{ w1_{\xi_n}, w2_{\xi_n}, \ldots wl_{\xi_n}$

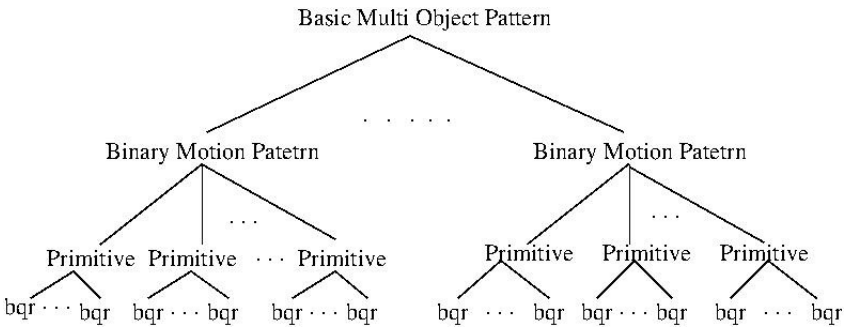

Figure 4.5: Hierarchical Organization of Motion Pattern

} }

Since the same *JEPD* sets of binary qualitative spatial relations are used for all the objects, the alphabet set Σ_{bmop} is same for all the ξ_is. Therefore, the strings in every set are made out of the terminals belonging to the same Σ_{bmop}. When L_{bmop} is defined as a set of sets, the inner curly braces demarcate the binary motion patterns for various primary-reference pairs.

The language L_{bmop} is regular because each string belonging to this language is generated by a regular grammar. After the introduction of basic multi object patterns, hierarchical organization of a motion pattern among multiple objects can be described as shown in Figure 4.5. In the figure, *bqr* stands for a value of a binary qualitative spatial relation. Each primitive pattern is a set of such values. A binary motion pattern **bmp** between two objects is formed using a set of primitive patterns. A basic multi object pattern **BMOP** among a number of objects is made up of a number of **bmp**s. The format of training data for basic multi object patterns is same as the one used for binary motion patterns. In the case of binary motion pattern, there is only Pattern_Specification field in each training record because only two objects participate and one of them is the reference. For basic multi object patterns, there may be multiple Pattern_Specification fields in a single training record. After learning it is necessary to represent a basic multi object pattern. Once learned, a basic multi object pattern can be represented in the following form:

```
Pattern_Name
Object_List
```

`Reference`

`Learned_Grammar`

`Temporal_Constraint_List`

`MaxHoldTime_bmop`

`MaxHoldTimeList_bmp`

The name of the basic multi object pattern along with the identities of the primary objects and the reference are stored in first three fields. The *Learned_Grammar* field stores the learned grammar for constituent binary motion patterns. Temporal constraints can be learned from training data. In a training record, the start point and end point are recorded along with string representation of a binary motion pattern. During processing of a training record, it is possible to compute Allen's interval algebra relations between pairs of binary motion patterns using the end point relations shown in Figure 4.4. Between the same two binary motion patterns, different Allen relations may hold in different training records. In *Temporal_Constraint_List*, the Allen relations between each pair of **bmp**s are maintained. The field *MaxHoldTime_bmop* stores the *HoldInterval* of the basic multi object pattern. The value of this field can be computed from the values of the constituent binary motion patterns during learning. The field *MaxHoldTimeList_bmp* is a list of values. Each value in this list gives the *HoldInterval* of a constituent binary motion pattern.

In our proposed framework, time is represented at the level of a basic multiobject pattern. An alternative will be to record time with each state i.e. with each primitive. When time is recorded with each state, it is possible to make a finer distinction between patterns using their temporal characteristics. On the other hand, two patterns, identical in terms of spatial characteristics, may not be considered identical if one takes a shorter/longer duration. For example, two overtake patterns may be identical in terms of spatial locations and directions; but when temporal duration is compared state wise, they may differ. In the thesis, we overlook this and have taken the view that if patterns are observed within the maximum learned interval for such patterns, then we check them for recognition.

4.5.2 Learning

For learning a basic multi object pattern, we need to perform two tasks. One is to learn the **bmp**s for each primary-reference pair. The other is to compute the Allen relations between *Hold Interval*s of binary motion patterns. The binary motion pattern between a primary object and a reference can be learned using the algorithm Learn_bmp_Modified. Each Allen relation between a pair of intervals can be expressed by relations between the end points of the intervals. In Figure 4.4, we have shown the definitions of Allen's interval algebra relations in terms of endpoints of the intervals [134]. Each interval is denoted by an ordered pair of the form (X_s, X_f). The first value in this ordered pair is the start point of the interval and the second value is the end point of the interval. The following algorithm learns a basic multi object pattern:

Algorithm 4 Learn_Basic_Multi(Training_Data)

Input: Training records

Output: A representation of the learned *bmop*

1: Learn_bmp_Modified(Training_Data)
2: Assign the learned grammar to the Learned_Grammar field
3: Find_Max_HoldTime_List_bmp(Training_Data)
4: Find_Max_HoldTime_bmop(Training_Data)
5: Find_Temporal_Constraint_List(Training_Data)

Complexity of Learning Basic Multi Object Pattern

In line 1 of algorithm 4, a call is made to *Learn_bmp_Modified*. For a particular application, the complexity of *Learn_bmp_Modified* is $O(L_P * L_P * N_T * N_T)$. Assigning the learned grammar involves scanning grammar productions and copying them into the *Learned_Grammar* field of the representation. This operation can be done in $O(L_P)$ time. The algorithm *Find_Max_HoldTime_List_bmp* computes maximum hold time for each binary motion pattern. Here, a scan through training records is necessary. During this scan, for each binary motion pattern, the minimum start time and the maximum finish time is computed. Therefore,

this algorithm actually computes the minimum and maximum among a set of time points. If N_O is the maximum number of objects in a training record, then *Find_Max_HoldTime_List_bmp* takes $O(N_T * N_O)$ time. At line 4, the maximum hold time for the basic multi object pattern is computed. Here, a scan is made through the *MaxHoldTimeList_bmp* field of the representation and the minimum start time and the maximum finish time among all **bmp**s are computed. Therefore, this operation takes $O(N_O)$ time. Temporal constraints between pairs of **bmp**s are computed at line 5 of the algorithm. Since Allen's interval algebra relations are closed under converse, it is not necessary to compute temporal constraints between all pairs of **bmp**s. For example, if the Allen relation between the first and the second **bmp** is computed to be p, then the relation between the second and the first is pi. Therefore, in a set of N_O **bmp**s, the Allen relation between the first and the remaining $N_O - 1$ are computed. Then, relations between the second and the remaining $N_O - 2$ are computed. Each computation involves four comparisons. So, the total number of comparisons is $O(4 * N_O * N_O)$. As there are N_T training records, the total time taken at line 5 is $O(N_T * N_O * N_O)$. Comparing this time with the time taken at line 1, we find that the parameter L_P dominates N_O because the maximum length of a binary motion pattern will far exceed the maximum number of objects in an application. Therefore, complexity of algorithm 4 is taken as O($L_P * L_P * N_T * N_T$).

4.5.3 Recognition of a Basic Multi Object Pattern

During recognition, we will have to compute primitive patterns between each primary-reference pair. New primitives occur when qualitative spatial relations between a primary-reference pair change. Before this recognition phase, one or more basic multi object patterns have been learned and represented. The Learned_Grammar field of each **BMOP** is a regular grammar. In algorithm 5, recogniser of a regular grammar i.e. a finite state automaton has been used for describing the recognition process. Whenever a new primitive is observed for an object, this primitive may trigger transitions in multiple learned automata. This is because of the fact that the object under consideration may participate

in multiple learned basic multi object patterns. Therefore, when a new primitive is seen for an object, it is necessary to find the learned **BMOP**s where this object participates. After identifying such **BMOP**s, we need to check whether this newly observed primitive triggers any state transitions. If so, then the state transitions are recorded in the **BMOP**s. This process continues as and when new primitive patterns occur in input. State transitions may stop because of two reasons. Firstly, a new primitive for an object may fail to trigger a transition because such a transition was not encountered during learning. Secondly, if elapsed time exceeds the maximum hold time of a basic multi object pattern, then it is not necessary to make state transitions in that **BMOP**. A learned **BMOP** is recognised if all of its constituent **bmp**s are recognised within the maximum hold time of the **BMOP**. In order to fulfill this requirement in terms of finite state automaton, at first we need to be in the start state of the automaton. This state corresponds to the start symbol of the learned grammar. From this start state, there is an edge to each state that corresponds to an *object non-terminal*. By *object non-terminal*, we mean a non-terminal in the grammar that corresponds to an object in the motion pattern. When a new primitive is seen for an object, a transition is attempted in the part of the automaton that starts with the non-terminal corresponding to the object under consideration. Then, for the **BMOP** to be recognised, the first requirement is that the automaton has to reach final states along each path starting with a *object non-terminal*. There are two other requirements. One is the about maximum hold time and the other is related with temporal constraints. During recognition, if the binary motion pattern for an object occurs for a duration that is more than the *MaximumHoldInterval* of the pattern, then we need to discontinue the process of recognising the **BMOP**. The other requirement is about satisfaction of temporal constraints by the binary motion patterns during recognition. When a constituent binary motion pattern is recognised, its start time and end time become known. Allen relations between these constituent **bmp**s can then be computed and checked for conformance with the ones stored during learning. The learned temporal constraints between each pair of constituent binary motion patterns need to be satisfied for recognition.

Algorithm 5 recognises a basic multi object pattern. In lines 1 to 3, we

set the status of each learned basic multi object pattern to a value termed as *Processing*. *Processing* status signifies that we have not yet recognised that **BMOP** and maximum hold time has not exceeded for this **BMOP**. The loop set up at line 1 that constantly checks whether any new terminal is seen for any of the objects participating in the pattern. Moreover, in this loop elapsed time is updated, violation of temporal constraints is checked and decisions regarding recognition of **BMOP** are taken. The steps from line 5 to line 33 handle new terminals seen in input. Whenever any new terminal is seen, it is necessary to check whether a transition is possible. Since the terminal is seen for a particular object, we need to examine each learned **BMOP** where this object participated. The for loop set up from line 6 to 32 takes care of this. Within this for loop, in line 7 and 13, we check whether the terminal is seen for the first time for the object. In that case, there is no possibility of discontinuity. We can make a transition if it is defined. In lines 14 to 31, we handle the situation where the terminal is not a first-seen one. In these lines, possibility of discontinuity is checked and transition is made if defined. Irrespective of the fact whether new terminal is seen or not, elapsed time is updated at line 34. In lines 35 to 53, various conditions of recognition and temporal constraint violation are checked. Conditions in lines 41 and 42 ensure that the algorithm terminates if some learned **BMOP** is recognised. Line 48 ensures that the algorithm terminates if no learned **BMOP** is recognised. Because of step at line 51, the algorithm will eventually terminate when maximum hold times of all learned **BMOP**s are exceeded.

Algorithm 5 Recognise_bmop

1: **for** each learned *bmop* **do**
2: Mark its status as *Processing*
3: **end for**
4: **while** true **do**
5: **if** there is a new terminal a_i for object O_j **then**
6: **for** each learned *bmop* **do**
7: **if** a_i is first seen for O_j and status of *bmop* is *Processing* **then**
8: **if** transition for O_j on a_i is defined **then**
9: Make a transition for O_j on a_i
10: Increment match count for O_j
11: **end if**
12: **end if**
13: **if** a_i is a neighbouring terminal of a_{i-1} and status of *bmop* is *Processing* **then**
14: **if** transition for O_j on a_i is defined following transition of a_{i-1} **then**
15: Make a transition for O_j on a_i
16: Increment match count for O_j
17: **end if**
18: **end if**
19: **if** a_i is not a neighbouring terminal of a_{i-1} and status of *bmop* is *Processing* **then**
20: $X \Leftarrow$ Find_Sequence(a_{i-1},a_i,Len)
21: **for** each terminal d in X **do**
22: Set Category of d to E, Set $Path\_Name$ of d to a_{i-1},a_i
23: **if** transition for O_j on d is defined **then**
24: Make a transition for object O_j on d
25: Increment match count for O_j
26: **end if**

```
27:                 end for
28:             end if
29:         end for
30:     end if
31:     Update_Time()
32:     if MaxHoldTime is exceeded for a bmp in some bmop then
33:         Make the status of corresponding bmop as NotRecognised
34:     end if
35:     if MaxHoldTime is exceeded for any learned bmop then
36:         Mark the status of that bmop as NotRecognised
37:     end if
38:     if all the constituent bmps have match count equal to their respective
    lengths then
39:         if all temporal constraints are satisfied for bmps then
40:             Mark the status of the bmop as Fully Recognised and Exit
41:         end if
42:     end if
43:     if Pattern string for each object in the new instance is processed then
44:         Output recognition status
45:         Exit
46:     end if
47:     if MaxHoldTime is exceeded for all the learned bmops then
48:         Exit
49:     end if
50: end while
```

Complexity of BMOP Recognition

In this complexity analysis, $*$ is the multiplication operator. We denote the number of learned **BMOP**s as N_B and the maximum path length encountered during discontinuity processing as L_M. Moreover, the maximum number of objects in any **BMOP** is assumed to be N_O. Then frequency of step 2 is $O(N_B)$. The for

loop in line 6 runs for $O(N_B)$ time. The frequency of the for loop in line 23 is $O(N_B * L_M)$. The parameter L_M is a constant for a particular application. This is so because L_M is the maximum path length encountered in discontinuity processing. Different events will have different values for this path length. The discontinuity processing algorithm finds the shortest paths from every node to every other node in a conceptual dependency graph (CNG). Out of these, the maximum path length is taken and used as the value of L_M. Since Conceptual Neighbourhood Graphs (CNGs) for a particular application are known (because the set of binary qualitative relations to be used are known), the value of L_M becomes constant for that application. Therefore, this frequency can be taken as $O(N_B)$. The step in line 35 runs for $O(N_B * N_O)$ time. The condition in line 38 can be checked in $O(N_B)$ time as each **BMOP** needs to be examined. The if condition in line 41 can be checked in $O(N_B * N_O)$ time. Temporal constraints are verified for each recognised **BMOP**. In section 4.5.2, it is shown that computation of temporal constraints takes $O(N_O * N_O)$ time. The steps 48 and 51 can be carried out in $O(N_B)$ time. In a real application, we assume that the number of learned **BMOP**s is more than the maximum number of objects participating in a pattern. With this assumption, the complexity of the algorithm can be taken as $O(N_B * N_O)$.

Illustrative Example

We consider a basic multi object pattern among four objects. The objects have identities A, B, C and D. Object D is the reference. This motion pattern is learned from a set of training records. For convenience of presentation, we choose to denote primitives by letters of alphabet. The training data set for this example is taken as:

```
Pattern_Name=Overtake
No_Of_Objects=4
Object_List=A;B:C
Reference=D
A=a,b,c,d;10;30
```

```
B=p,q,r,s:20;40
C=u,v,w,x:5;40

Pattern_Name=Overtake
No_Of_Objects=4
Object_List=A;B;C
Reference=D
A=a,b,c,d;25;50
B=p,q,t,s:20;30
C=u,v,w,x:15;20

Pattern_Name=Overtake
No_Of_Objects=4
Object_List=A;B;C
Reference=D
A=a,g,c,d;10;45
B=p,q,r,s:15;40
C=y,v,w,x:4;40
```

When this *bmop* is learned, its representation will look like:

Pattern_Name=Overtake

Object_List=A;B;C;D

Reference=D

Learned_Grammar= $S \rightarrow A \mid B \mid C$, $A \rightarrow a\ A_1$, $A_1 \rightarrow b\ A_2 \mid g\ A_2$, $A_2 \rightarrow c\ A_3$, $A_3 \rightarrow d\ A_4$, $A_4 \rightarrow \epsilon$

$B \rightarrow p\ A_5$, $A_5 \rightarrow q\ A_6$, $A_6 \rightarrow r\ A_7 \mid t\ A_7$, $A_7 \rightarrow s\ A_8$, $A_8 \rightarrow \epsilon$, $C \rightarrow u\ A_9 \mid y\ A_9$, $A_9 \rightarrow v\ A_{10}$, $A_{10} \rightarrow w\ A_{11}$, $A_{11} \rightarrow x\ A_{12}$, $A_{12} \rightarrow \epsilon$

Temporal_Constraint_List=(A *o* B or A *di* B);(A *d* C or A *pi* C or A *o* C);(B *f* C or B *pi* C)

MaxHoldTime_bmop=(4,50)

MaxHoldTimeList_bmp=(A,10,50);(B,15,40);(C,4,40)

Temporal_Constraint_List field enumerates the temporal constraints between pairs of binary motion patterns. A binary motion pattern in this field is specified by the identity of the object for which the pattern is defined. Since, Allen's each Allen algebra relation has a converse, it is not not necessary to enumerate all possible combinations. For example, if the Allen relation between A and B is stored, the relation between B and A can be found by taking its converse. In the *MaxHoldTimeList_bmp* field, the maximum hold time for each constituent binary motion pattern is listed. This can be found from the training records by considering the earliest start time and latest finish time for a binary pattern. The *MaxHoldTime_bmop* field stores the maximum hold time for the basic multi object pattern. In the example representation, the earliest start time for any constituent **bmp** is 4 and latest finish time for any constituent **bmp** is 50. Therefore, Maximum hold time for the *bmop* is *(4,50)*.

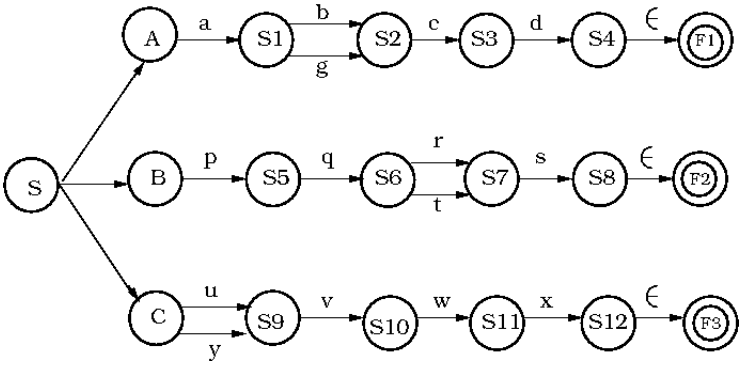

Figure 4.6: Representation of bmop using FSA

In Figure 4.6, the learned grammar is represented by a finite state automaton. The initial state is S and it corresponds to the start symbol of the grammar. In order to explain how recognition is done using algorithm 5, we assume that input is in the following form:

```
A=a,g,c,d;10,30
B=p,q,r,s;15,40
C=y,v,w,x;15,40
```

In each line, the sequence of primitives for the binary motion pattern for an object is listed. The hold interval for the **bmp** is also shown. In this example, there is only one learned basic multi object pattern. New primitives, observed for objects A, B and C, will trigger state transitions in the automaton. For example, when the primitive a is seen for object A, a transition will be made from state S to state A and then to state *S1*. When primitive p is observed for object B, transition will be made from state S to state B and then to state $S5$. For the example input, transitions will eventually take us to the final states *F1*, *F2* and *F3*. This means that each binary motion pattern corresponding to an object is recognised. Once the final states are reached, temporal constraints between the input **bmp**s can be computed. The temporal constraints are:

```
(A o B), (A o C), (B eq C)
```

The observed temporal constraints are in conformance with the ones computed during learning. The hold time for the input *bmop* is *(10,40)* i.e. the duration of the observed basic multi object pattern is 30. This duration does not exceed the *MaxHoldTime* of the learned *bmop*. Therefore, the input basic multi object pattern is recognised by algorithm 5.

In the next chapter, we present design of a qualitative description language for representation and recognition of motion patterns. Binary motion patterns, introduced in this chapter, can represent motion between two objects only. The description language, proposed in the next chapter, can express motion pattern among multiple objects. Moreover, using this language, motion patterns can be constructed hierarchically and temporal constraints among sub-patterns can be expressed.