

TABLE 4.2  
NET CUT SAMPLE

| Notes:  | Cut CP File Example   |
|---|---|
| Via coordinates adjusted to offset points.            | D 4110 1970 * CW1W2 0 80 80<br>D 3570 1970 * CW1W2 0 80 80<br>W 3500 1900 4200 3000 Interrupt |
| Pin numbers for first wire of net remain constant.    | S 80 V Metal I Interrupt 10 P 1<br>S 80 V Metal I Interrupt 11 P 2<br>S 90 H Metal P 1 P 2    |
| Complimentary wire of net.                            | D 4170 2030 * CW1W2 0 80 80<br>D 3630 2030 * CW1W2 0 80 80                                    |
| W line duplicated.                                    | W 3500 1900 4200 3000 Interrupt   |
| Pin numbers offset to correspond to D line additions. | S 80 V Metal I Interrupt 12 P 3<br>S 80 V Metal I Interrupt 13 P 4<br>S 90 H Metal P 3 P 4    |

A partially correct solution was presented. The errors generated manifested themselves in subsequent nets of the routing solution where changes had not been made. An examination of the particulars identified the subtle difficulty. Pin numbers in the cp file have several dependencies. They correspond in a one to one fashion with the D-lines in the file. Also, within a net definition, they can occur in any sequence, but still refer back to the appropriate D-line of that net definition. By introducing an additional net, there were now

two additional D-lines in the file. Since this net occurred in the middle of the file, the D-lines added were in the interior of the file. Consequently, the pin numbers of all nets following the one where the split was created by duplicating the net entries, were no longer in synchronization. Pin numbers in one net structure now related incorrectly to D-lines in another net structure. This realization pointed out the intricate interaction among cp file entries.

By making the necessary modifications to adjust all subsequent pin numbers, the cp file was again displayed. At this juncture, the fat wire routing solution was depicted, with the "cut" net properly portrayed as two separate nets, with all segments constructed in the fabrication size wire geometry. This small test indicated that the overall concept was sound, but that the implementation details were even more complex than originally anticipated.

#### **4.2.2 CAPTURING THE IDEAS**

Once it had been demonstrated that cutting could be accomplished through cp file modifications, the focus shifted to determining whether or not nets of this type could be identified and split in an automated fashion. Clearly, the place to begin was to capture the steps necessary to properly bifurcate the simple category of three segment nets that had been used in the proof of concept example. Reviewing the manual operations, it was determined that the steps could be automated for all nets of that type (neglecting cell reflections and inversions at this point), providing nets of that topology could be identified in the cp file.

A visual analysis of the cp file revealed that the simple topology net seemed to be easily identified by its characteristics: (1) two via or D-lines, (2) three segment lines, (3) two of the segment lines had instances (cell ports) at one end and vias at the other, and (4) one horizontal segment with vias at each end. If the cp file could be scanned, and nets with

---



these characteristics tagged, the cutting steps learned from the first example could be applied automatically. This would, in effect, accomplish the preliminary cutting of all nets of this type.

A small filter program was constructed. Its sole purpose was to scan the cp file and attempt to identify nets in the primitive net category, using the constraints listed above. This technique worked, and all of the nets conforming to these conditions were detected. Additionally, all were recognized in a single pass over the components of each net. All constraint checking could be performed with basic math and comparison functions.

### **4.3 EXTENDING THE TECHNIQUE**

The next step was to extend the technique to other net types. The logical extension from the two instance net was a net with three instance segments. Another detection program was developed which operated in a manner similar to the previous, only varying the recognition criteria to look for three instance segments instead of two. The initial run of the modified classifier appeared to work perfectly. A more detailed examination of the results turned up a flaw in the tests.

The enthusiasm associated with the previous success clouded some of the critical thinking and analysis in rewriting the select criteria for the new net type. The conditions were modified to account for the extra segment and instance to a port, but the total via criteria was omitted. This was based on the assumption that a via would exist for each instance segment. Although the recognizer returned all of the nets types for which it was designed, it also returned nets having three instance segments but only two vias. An example of such a net is provided in Fig. 4.3



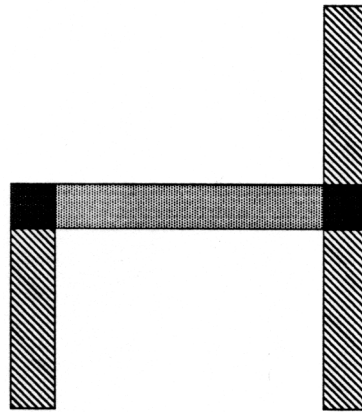


FIG. 4.3 THREE INSTANCE TWO VIA NET

This failure served to bring an important point to light. There had to be some form of verification mechanism which guaranteed correctness of the scanning criteria used to identify the type of net being sought. However, it would take the development of the finite state machine theory and regular expression link of chapter five to provide this verification.

This incremental extension approach continued, gradually adding net variations to the set of net types that could be recognized. Mid way through this undertaking, it was noted that the recognition criteria were very similar in structure, with almost identical components being tallied in one manner or other. The differences between nets could be uncovered in the counts and relationships among the components. The identification operation had really transformed into a pattern matching problem, and the idea of using the computer vision idea of feature vectors was put in place.

## 4.4 FEATURE VECTORS

The first feature vectors consisted of only a limited set of components. These were: (1) the number of vias, (2) the number of instance (port) segments, (3) the total



number of segments, and (4) the number of connectors. This early vector,  $F_i$ , was then defined as:

$$F_i = \{ \begin{array}{l} \text{number of vias,} \\ \text{number of ports,} \\ \text{number of connectors,} \\ \text{number of segments,} \\ \text{(constraints)} \end{array} \}$$

Experience demonstrated that this relatively simple vector could accurately identify most of the simple net categories. However, as more and more complex nets were encountered, the vector components were modified to account for the concept of layers, or nets possessing multiple backbones.

This revised set of vectors,  $F_i$ , is composed of components reflecting counts of the primary net elements: (1) number of vias, (2) number of ports, (3) number of connectors broken into type by East/West (E/W) and North/South (N/S), and (4) the number of backbones or layers. Each of these characteristics can be counted in time proportional to one scan over the components of the net. The vectors now appeared as:

$$F_i = \{ \begin{array}{l} \text{number of vias,} \\ \text{number of ports,} \\ \text{number of connectors(E/W),} \\ \text{number of connectors(N/S),} \\ \text{number of segments,} \\ \text{number of layers(backbones),} \\ \text{ (constraints)} \end{array} \}$$

The value of  $i$  ranges from 1 to  $R$ , where  $R$  represents the number of net categories. By examining the relationship among the components of the vector, the desired categorization can be accomplished.

Rewriting the vector components so that the number of vias is  $X_1$ , the number of ports is  $X_2$ , the number of E/W connectors is  $X_3$ , the number of N/S connectors is  $X_4$ , and number of layers or backbones is  $X_5$ ,  $F_i$  can be written as

$$F_i: \{ X_1, X_2, X_3, X_4, X_5 \mid (\text{constraints}) \}$$

where *constraints* represents the conditions imposed on the relations among the vector components. For a Category 2 type net, the vector  $F_2$  with explicit conditions is defined as-

$$F_2: \{ X_1, X_2, X_3, X_4, X_5 \mid X_1 == X_2, X_3 == 0, X_4 == 0, X_5 == 1 \}$$

For the net portrayed in Fig. 4.4, its feature vector,  $F_{observed}$ , can be constructed by tallying components during a scan of the net. The vector would appear as follows:

$$F_{observed}: \{ X_1 == 4, X_2 == 4, X_3 == 0, X_4 == 0, X_5 == 1 \}$$

Since the relationship among the components of  $F_{observed}$  satisfy the conditions of  $F_2$ , the net can be categorized as a Type 2 net and a specialized routine invoked to make the proper modifications to the cp file to effect the splitting.

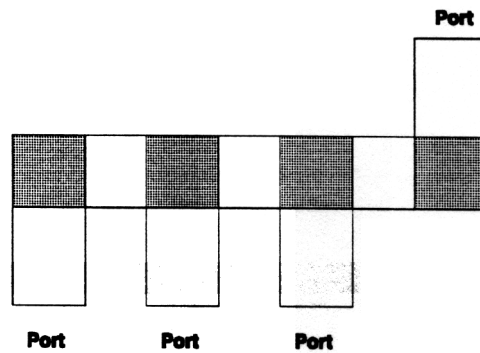


FIG. 4.4 CATEGORY 2 SLE NET

Applying the feature vector idea in this way carried many advantages. For typical image processing applications, the vector components consist of either probability values or some real value approximation. In this problem domain, the vector components were discretized and unique. As a result, the evaluation function for deciding which among the

competing feature vectors is the best, is a simple true or false decision, not a computationally intensive combinatoric function. Second, it allows the net categories to be specified in a pseudo-mathematical manner. This is especially important when the finite state machine theory is applied in chapter five. Finally, as mentioned earlier, the component counts can be summarized and evaluated during a single pass over the cp file. This can be accomplished in linear time.

As the feature vectors evolved, the constraint equations changed in incremental ways. By observing this evolutionary transform of the vector conditions, and the steps involved in “splitting” the cp file for each type of net, it became obvious that not all variations were different types of nets. Instead, they were viewed as variations within a larger category. It was this realization that triggered the development of a net taxonomy tree.

## 4.5 NET TAXONOMY

As each category of net was identified, a node for a taxonomy tree was set aside. At the outset, the structure of the tree was not obvious. Only as the synthesis mentioned above was carried out, did the true structure of the tree reveal itself.







The steps necessary to split the first sample nets classified, Fig. 4.2, turned out to be identical to those for the nets with an equal number of additional port segments and vias. So rather than having a category for nets with every possible number of port segments and vias, the category was expanded to incorporate all who had a single layer, no connectors, and number of vias equal to the number of port segments. This became one cluster node in the tree.

---



Before proceeding further with the taxonomy discussion, where many net example figures will be necessary for understanding, a simplified net symbology is presented. Table 4.4 provides the legend.

TABLE 4.4  
NET SYMBOL TABLE

| Symbol   | Meaning               | Symbol  | Meaning             |
|--|-----------------------|---|---------------------|
|   | Via                   |    | East/West Connector |
|   | Instance (Port)       |    | Vertical Segment    |
|  | North/South Connector |  | Horizontal Segment  |

A representative sample of nets from the cluster node just described appears in Fig. 4.5. If the vector components are totaled for each of these nets and an  $F_{\text{observed}}$  generated, the constraint conditions for a Type 2 net will be satisfied. One can see that in this way, (1) a net can be scanned in linear time, (2) the  $F_{\text{observed}}$  vector can be generated through accumulator counts of components, and (3) the constraint conditions checked through simple conditional tests. All of these are linear time operations. The overall result is a linear time categorization of fat wire nets through the use of feature vectors. Once categorized, specialized routines designed to perform appropriate cp file manipulations are invoked.

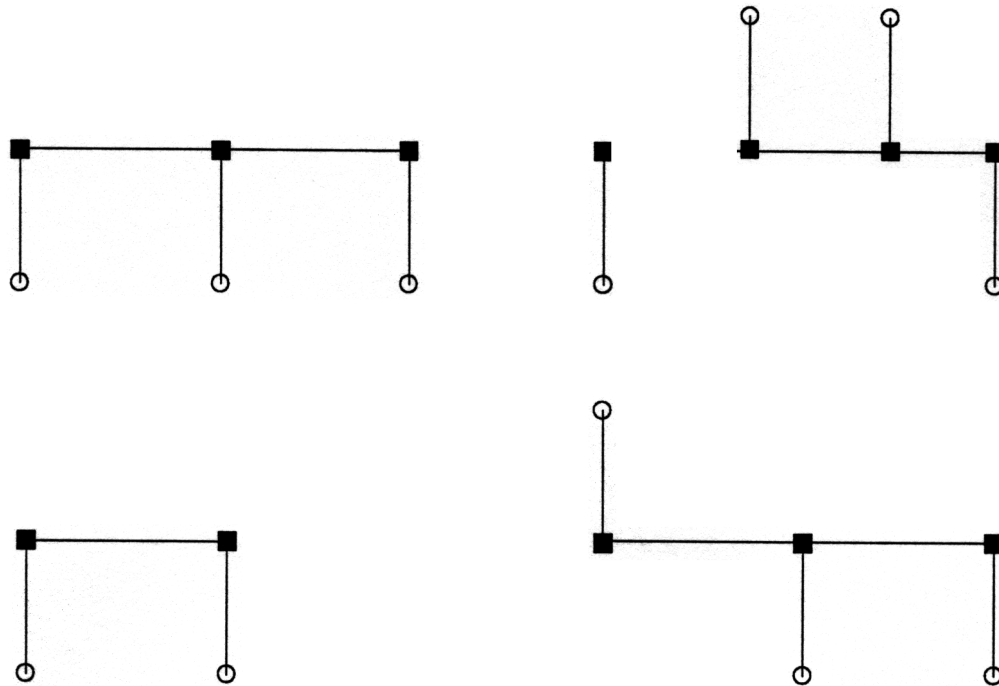


FIG. 4.5 REPRESENTATIVE TYPE 2 NETS

The first indication of equivalence among net configurations came from comparisons of steps required to bifurcate the fat wire net. When these steps coincided with other net topologies, as is the case of the original sample net of Fig. 4.2 and the other nets depicted in Fig. 4.5, the nets appeared to form a class. This entire class was described by the feature vector constraints associated with it. As this realization set in, the ability to actually bifurcate the nets of each class had to be considered. This led directly to the finite state machine theory, which not only proved equivalence of nets within a class, but at the same time guarantees that the nets of the class are bifurcatable.

Although the finite state machine theory developed in parallel with the net taxonomy, it is presented in a sequential fashion in chapter five of the dissertation for improved clarity and understanding. But as the remainder of the taxonomy discussion

progresses, it should be kept in mind that as the taxonomy classes were formed it is implied that nets in that class are bifurcatable.

#### 4.4.1 THE PRIMITIVE NETS

Net categories were added incrementally. Each new category added was based on similarities with a previous category, usually coupled with one subtle change or modification. After the first category of nets (Type 2: Single Backbone, number of Vias equal to the number of Port Segments, and no Connectors) was formed, the logical extension was to form a category with similar characteristics, but possessing a single East/West Connector. The modification to the feature vector constraints was minimal: only adding the condition that the number of E/W Connectors be equal to one. A representative sample of such configurations based on extensions of the Type 2 nets is provided in Fig. 4.6.

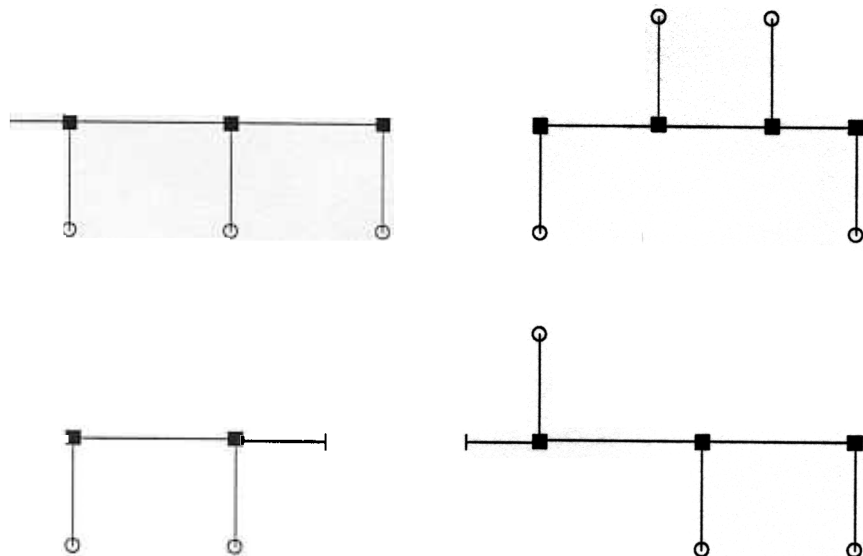


FIG. 4.6 REPRESENTATIVE TYPE 3 NETS



From the single E/W Connector, the next category formed contained nets with the basic structure of the Type 2 nets, and a single North/South Connector. For this class of net the feature vector constraint list changed somewhat more than for a Type 3. The requirement to have at least one via for each port segment remained, but an additional via now became necessary to attach the N/S Connector segment. The stipulation is fully explained in chapter 6, when the bifurcation of over-constrained vias is discussed. The revised vector for Type 4 nets appears as:

$$F_4 : \{X_1, X_2, X_3, X_4, X_5 \mid (\text{conditions}) \}$$

where the conditions are:

$$X_1 = X_2 + 1$$

$$X_3 = 0$$

$$X_4 = 1$$

$$X_5 = 1$$

The other fundamental category that emerged early in the research was the single segment net connecting a port to a North/South Connector. These occurred quite frequently along the north and south periphery of standard cell areas.

#### 4.4.2 MULTIPLE BACKBONE NETS

With these four categories a large percentage of nets could be bifurcated. The early taxonomy linking the four categories of nets from a root node of bifurcatable nets is shown in Fig. 4.7. The next class of nets to be added moved into the multiple backbone arena.

A multiple backbone net was any net that had horizontal segments not all of which existed on a given horizontal track. Thus, a multiple backbone net could be a short net which just happened to include a vertical jog from one routing track to the next enroute from source to destination. Or, it could be a relatively complex net, using feed through

passages in standard cell rows to migrate to a completely different channel or channels. So long as the vertical segment forming the connection between layers had a via at each end, and those vias were not over-constrained, the splitting techniques used for the four types already identified could be extended.

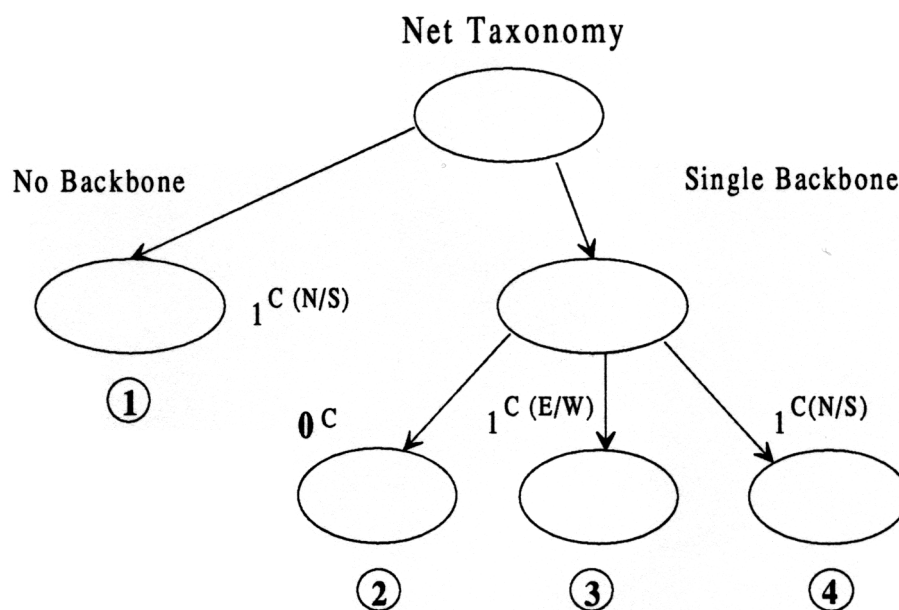


FIG. 4.7 FIRST TAXONOMY TREE

As the multiple backbone analysis continued, similarities began to appear between the primitive four categories and the partial net components that remained if the connecting vertical link between backbone layers was broken. An example of applying a “cut” to a multiple layer net is given in Fig. 4.8. The cut line partitions the net into sub-nets, each of which is now a single backbone net. Since the connecting vertical segment can be bifurcated without interfering with the normal polarity propagations, conceptually it can be removed. Once removed, the two sub-nets can be “spliced” together. This action reduces the multiple backbone analysis for this particular net back to a single backbone version. An examination of the feature vector for the resulting spliced net reveals a Type 3. Employing the techniques developed to bifurcate a Type 3 net, and adding the

operations necessary to bifurcate the vertical joining segment, this new type net can be appropriately dealt with. Since the joining segment only connects two backbone layers, and the vias at either end are dedicated strictly to the vertical segment, it is referred to as an under-constrained linkage. This is due to the fact that each via introduces a degree of freedom into the polarity propagation problem. With two vias, and only one segment on which to resolve polarity, an additional degree of freedom is still available. These ideas are fully discussed in chapter six.

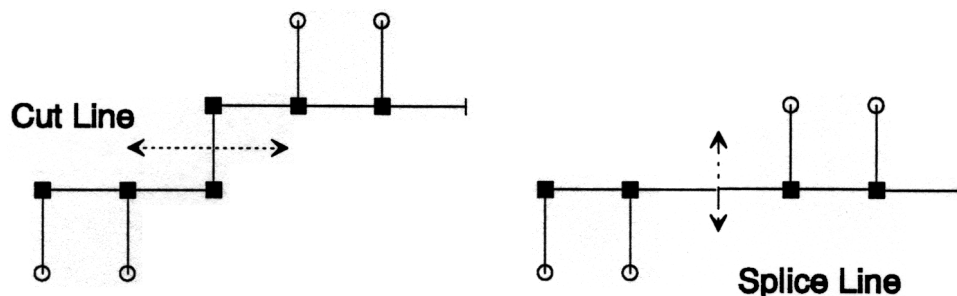


FIG. 4.8 MULTI-BACKBONE SPLICE. (a) ORIGINAL SLE WITH CUT LINE,  
(b) SLE WITH SPLICE

With the realization that a conceptual cut line approach would dramatically simplify the bifurcation of complex, multiple layer nets, the net taxonomy structure began to take on a more logical form. From a generic root node leading to each category that had been identified, the root now had leaves corresponding to either no backbone, a single backbone, or multiple backbones. (Where the term multiple backbones encompasses not only nets linked on two layers, but any number of layers.) Then, under the single backbone node, the three nodes corresponding to no connectors, an E/W connector, and a N/S connector were attached. These three represented all of the primitive types except for the single port segment to a N/S Connector, which resided under the no backbone leaf node.



This modified tree is depicted in Fig. 4.9. Each of the original four types of nets are labeled with numbered circles indicating the node into which they now fall.

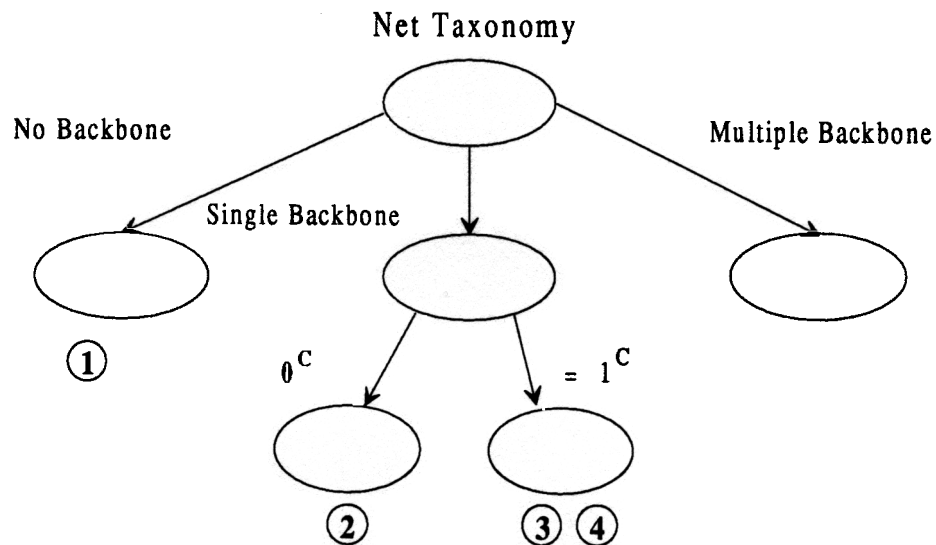


FIG. 4.9 REVISED TAXONOMY TREE

The final multiple layer type identified was named the fully constrained category. The name is derived from the fact that the vertical linking segment joins multiple layers together. An example is provided in Fig. 4.10. It is easy to see that sufficient freedom exists at the two end vias adjoining the top-most and bottom-most sub-nets, to permit proper segment bifurcation. However, at the intermediate vias, the freedom

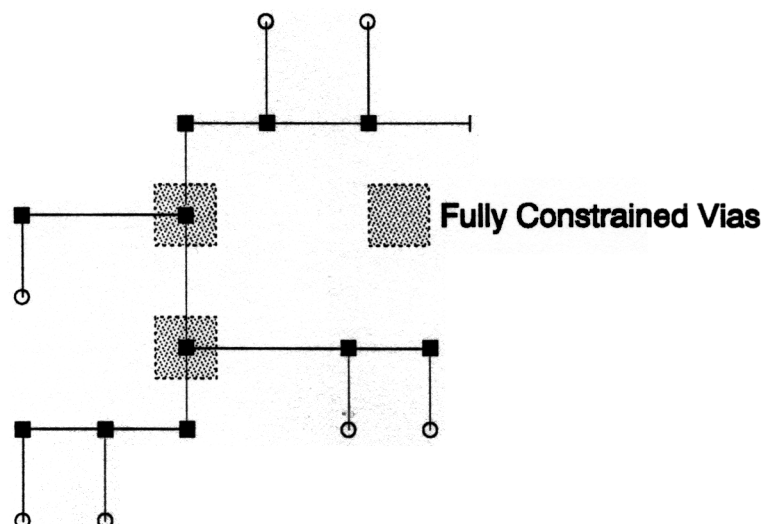


FIG. 4.10 MULTIPLE LAYER SLE NET WITH FULLY CONSTRAINED VIAS

is eclipsed once the end via polarities are established. Those intermediate vias are fully constrained by the backbone polarity of the sub-net that must be attached to the vertical joining segment.

Once again the "cut line" and "splice" approach used for simple multiple backbone nets was found to be valid in performing the conceptual analysis. Applying the cut lines to the net in Fig. 4.10, produces the pseudo-net of Fig. 4.11. Then, splicing the individual sub-nets together reveals the fact that the underlying configuration coincides with that of a Type 3 net, and is depicted in Fig. 4.12.

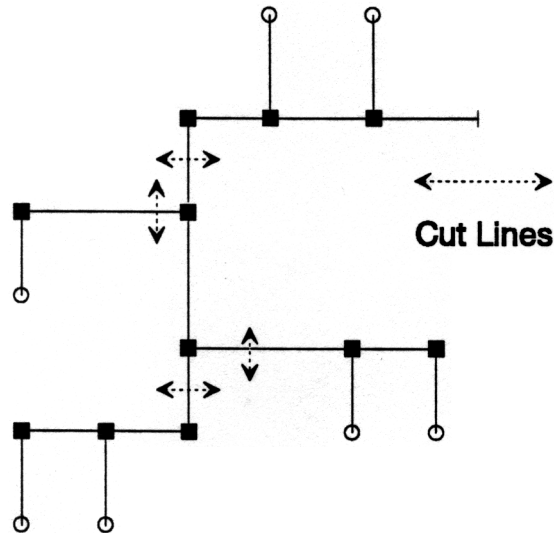


FIG. 4.11 FULLY CONSTRAINED NET WITH CUT LINES

#### 4.4.3 THE FINAL TAXONOMY

Following through with the analysis for simple multiple backbone nets and then for the fully constrained multiple backbone nets, the true underlying structure of the taxonomy tree was found. The root node of the tree represents the complete set of all forms of readily bifurcatable nets. From the root node, the first major discriminator is the number and type of backbones. There are the simple, single vertical port segment to a N/S

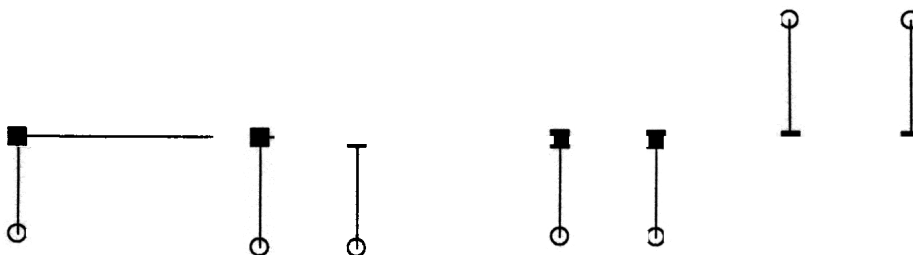


FIG. 4.12 SPLICED COMPONENTS OF FULLY CONSTRAINED NET

Connector nets that have no backbone. Then comes the node containing all nets with a single backbone. Finally, the third child underneath the root node represents the set of all multiple backbone nets.



The multiple backbone node can then be partitioned into two clusters. First, is the under constrained net class, which represent the traditional multiple backbone nets with a vertical segment with two vias joining the layers. The other class, the fully constrained, consists of those multiple backbone nets where the vertical joining segment connects more than two layers.

From this level in the tree, a symmetry emerges with respect to the follow on categorizers. From each node, the test for the presence of a connector provides the next level of discrimination. Under the stem containing connectors, the natural partition between E/W and N/S occurs. Using this partitioning methodology, I arrived at the final taxonomy tree.

Ongoing study throughout the research prompted several small changes. When connectors in a net were considered, it was initially assumed that there would be only one. However, as chips with more complex standard cell areas were encountered, it was discovered that nets with multiple connectors were not just an anomaly, but existed for a purpose. This prompted a reexamination of the tree structure. From the feature vector approach, and supported by minor changes in the taxonomy tree, the restriction to a single connector on a net was removed. Additionally, the possibility of having both a N/S and an E/W connector was identified and taken into account. The final result of this effort produced the tree in Fig. 4.13.

The final taxonomy tree resulted in thirteen net categories. The primitive nets, categories one through five, can be viewed as building blocks. By connecting various ones with vertical segments, the multiple backbone configurations can all be generated.

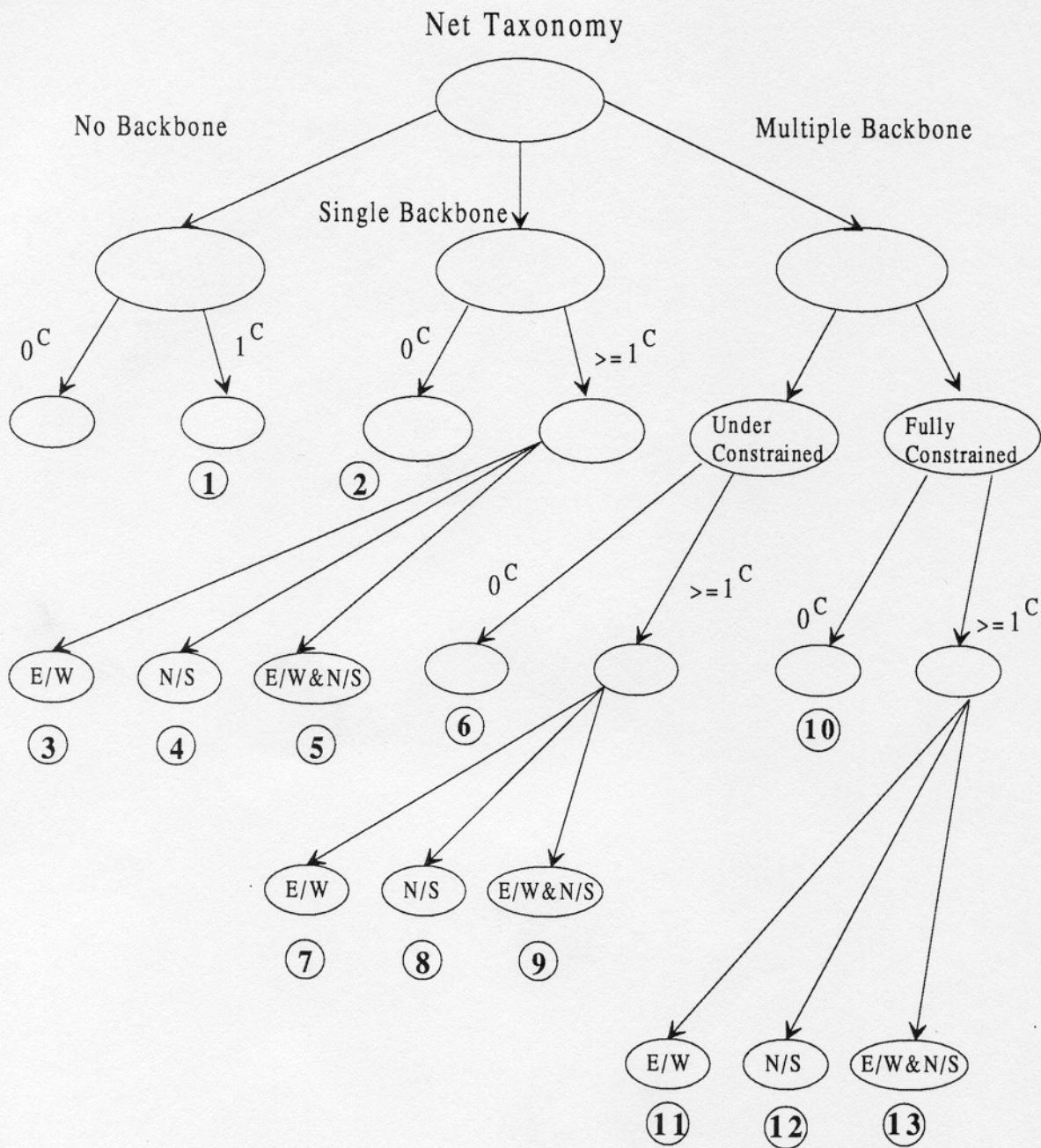


FIG. 4.13 FULL NET TAXONOMY

## 4.6 FEATURE VECTOR DECISION SIEVE

One of the advantages of the feature vectors that was mentioned earlier, was the fact that the vector components could be collected and tallied in one scan over the net. This clearly can be completed in linear time. But with thirteen possible vector matches, testing the conditional combinations for each to determine the correct vector match would still be linear time, but carry a large constant value  $C$  when examined using Big O notation.

Since the taxonomy tree evolved only after many of the net classes had been compartmentalized, it was not immediately obvious how the tree structure might help. Concurrently, with the development of net classes, system code was being generated to incorporate new vectors as their characteristics were identified. It was not until the time complexity analysis was thoroughly reviewed that an alternative emerged to reduce the vector decision time constant.

The technique used in the code implementation was to scan each net using each vector criteria. At the outset, this seemed reasonable, since only a handful of net classes had been identified. However, as the number of classes grew, the drawback associated with testing the conditionals for each vector independently also grew. Reviewing the code revealed that the same or similar component check was being run repeatedly, since many of the vectors had conditions that only differed in one field. Taking this observation and reviewing the final taxonomy tree produced an elegant solution.

Rather than test all conditions for all vectors, and then decide based on the match results, a decision flow could be set up. Paralleling the concept used in the Sieve of Eratosthenes for generating/testing for prime numbers, the vector conditions could be examined using a taxonomy sieve. The component tests would be ordered just as the discriminators in the taxonomy tree. Number of backbones, or layers would be the first

condition tested. That result would trigger the test at the next subsequent node. Testing component conditions would continue until arriving at a leaf node, at which time the vector would be identified by the node itself. This node type would also describe the net type. Once identified, the specialized splitting routines could then be invoked.

This sieve methodology cuts a significant amount of time from the vector identification process. Instead of being a function of number of vectors times the constraints per vector, it can now be performed in time proportional to the height of the taxonomy tree.

The current operational code was developed in a research setting and is still structured in the original, less efficient form. At the point that the development version is converted to production form for long term maintainability, this vector recognition technique should be incorporated.

## **4.7 FEATURE VECTOR COMPENDIUM**

The following 13 pages provide a collection of the thirteen feature vectors, along with their specific constraints. Several representative sample nets corresponding to the category recognized by the vector will be provided on the same page. The vector components underwent one final modification after the taxonomy tree was finalized. A count of fully constrained vias is maintained. The revised generic component list is presented with feature vector one.

---

### 4.7.1 CATEGORY 1 FEATURE VECTOR

The feature vector for Type 1 nets follows:

|            |                                   |    |
|------------|-----------------------------------|----|
| $F_1 = \{$ | number of vias,                   | X1 |
|            | number of port segments,          | X2 |
|            | number of connectors(E/W),        | X3 |
|            | number of connectors(N/S),        | X4 |
|            | number of layers(backbones),      | X5 |
|            | number of fully constrained vias, | X6 |
|            | (conditions)                      |    |
| $\}$       |                                   |    |

where the conditions are:

$X_1 == 1;$   
 $X_2 == 1;$   
 $X_3 == 0;$   
 $X_4 == 1;$   
 $X_5 == 0;$   
 $X_6 == 0;$

Sample nets of Type 1 are shown in Fig. 4.14.

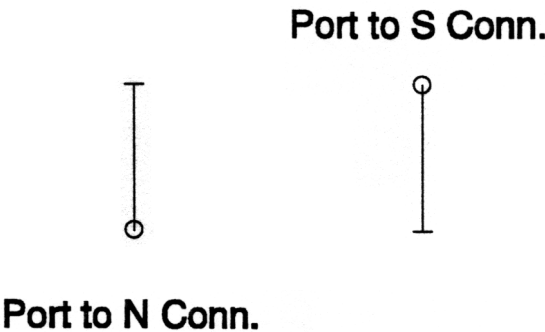


FIG. 4.14 TYPE 1 NET SAMPLES

#### 4.7.2 CATEGORY 2 FEATURE VECTOR

The feature vector for Type 2 nets follows:

$$F_2 : \{ X1, X2, X3, X4, X5, X6 \mid (\text{conditions}) \}$$

where the conditions are:

$$X1 == X2;$$

$$X3 == 0;$$

$$X4 == 0;$$

$$X5 == 1;$$

$$X6 == 0;$$

Sample nets of Type 2 are shown in Fig. 4.15.

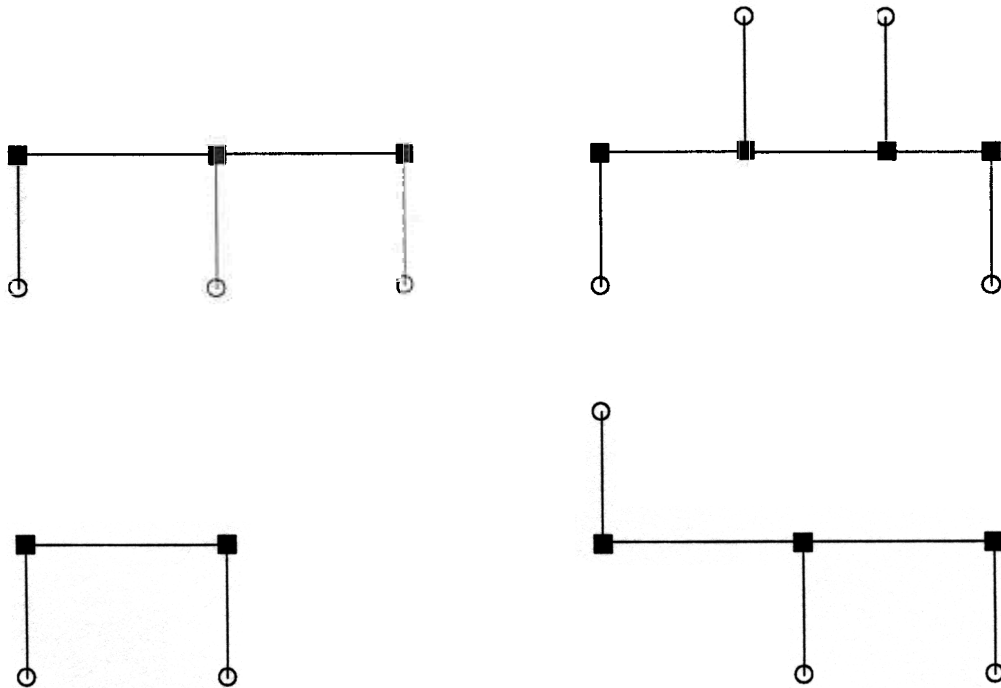


FIG. 4.15 TYPE 2 NET SAMPLES

### 4.7.3 CATEGORY 3 FEATURE VECTOR

The feature vector for Type 3 nets follows:

$$F_3 : \{ X1, X2, X3, X4, X5, X6 \mid (\text{conditions}) \}$$

where the conditions are:

$$X1 == X2;$$

$$X3 \geq 1;$$

$$X4 == 0;$$

$$X5 == 1;$$

$$X6 == 0;$$

Sample nets of Type 3 are shown in Fig. 4.16.

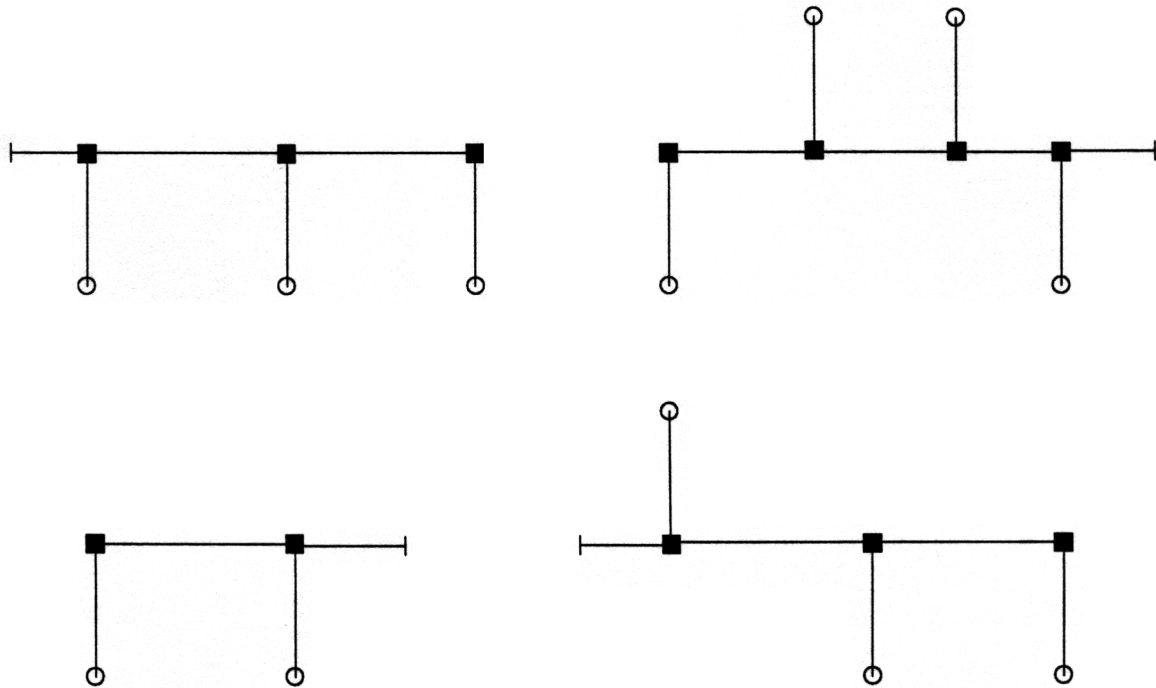


FIG. 4.16 TYPE 3 NET SAMPLES

#### 4.7.4 CATEGORY 4 FEATURE VECTOR

The feature vector for Type 4 nets follows:

$$F_4 : \{ X_1, X_2, X_3, X_4, X_5, X_6 \mid (\text{conditions}) \}$$

where the conditions are:

$$X_1 == X_2 + X_4;$$

$$X_3 == 0;$$

$$X_4 \geq 1;$$

$$X_5 == 1;$$

$$X_6 == 0;$$

Sample nets of Type 4 are shown in Fig. 4.17.

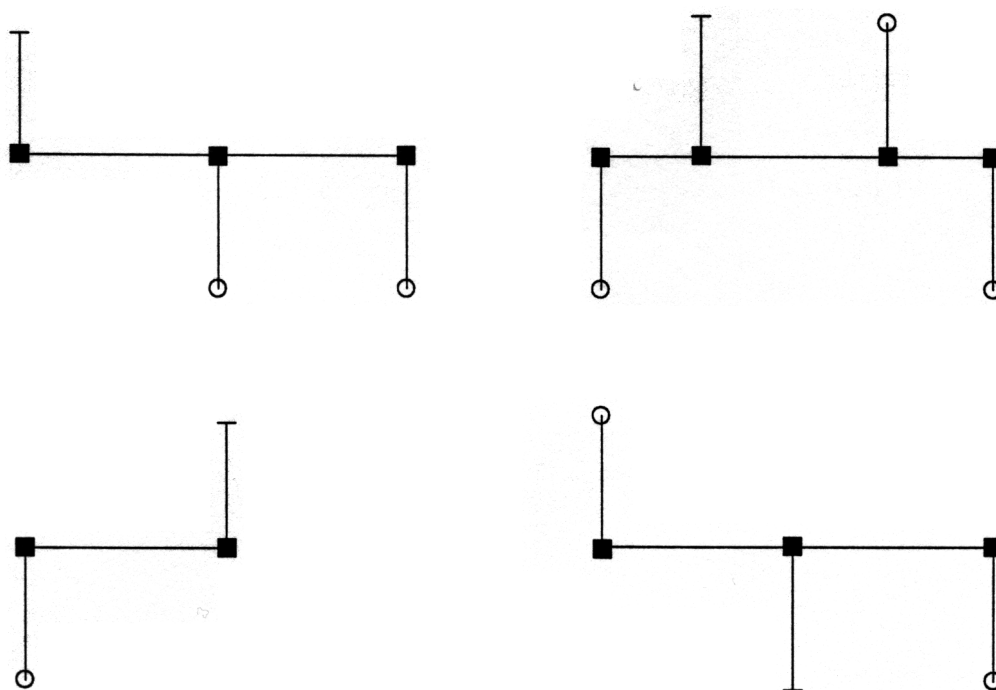


FIG. 4.17 TYPE 4 NET SAMPLES



#### 4.7.5 CATEGORY 5 FEATURE VECTOR

The feature vector for Type 5 nets follows:

$$F_5 : \{ X_1, X_2, X_3, X_4, X_5, X_6 \mid (\text{conditions}) \}$$

where the conditions are:

$$X_1 == X_2 + X_4;$$

$$X_3 \geq 1;$$

$$X_4 \geq 1;$$

$$X_5 == 1;$$

$$X_6 == 0;$$

Sample nets of Type 5 are shown in Fig. 4.18.

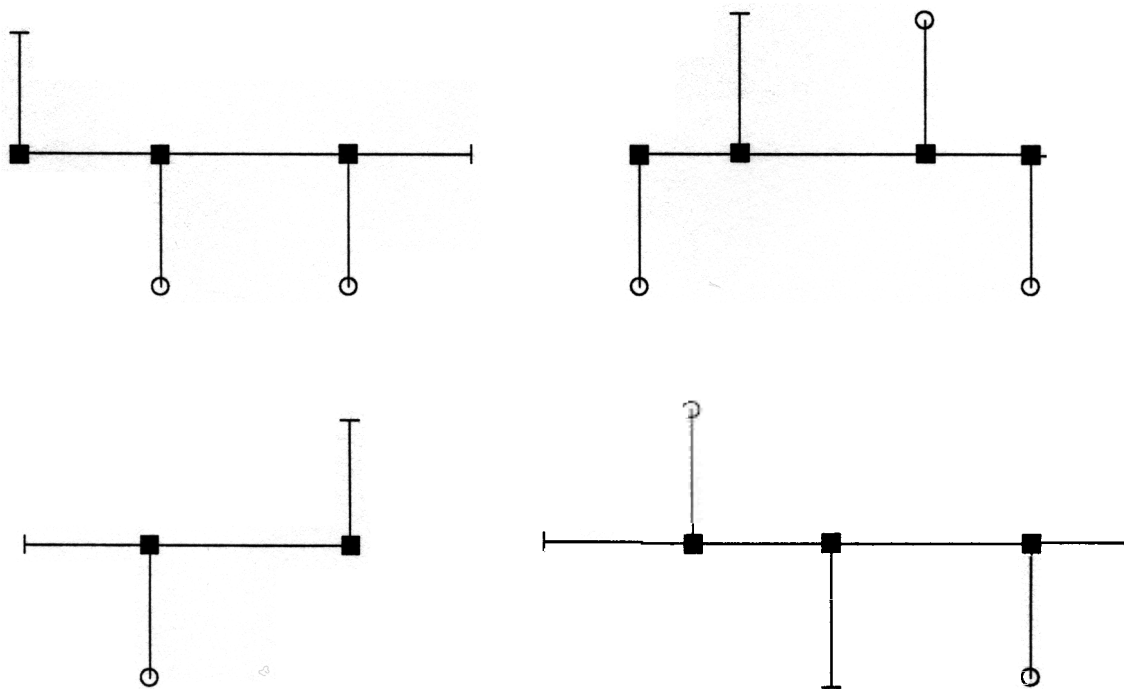


FIG. 4.18 TYPE 5 NET SAMPLES

#### 4.7.6 CATEGORY 6 FEATURE VECTOR

The feature vector for Type 6 nets follows:

$$F_6 : \{ X_1, X_2, X_3, X_4, X_5, X_6 \mid (\text{conditions}) \}$$

where the conditions are:

$$X_2 == X_1 - [(X_5 - 1) * 2];$$

$$X_3 == 0;$$

$$X_4 == 0;$$

$$X_5 > 1;$$

$$X_6 == 0;$$

Sample nets of Type 6 are shown in Fig. 4.19.

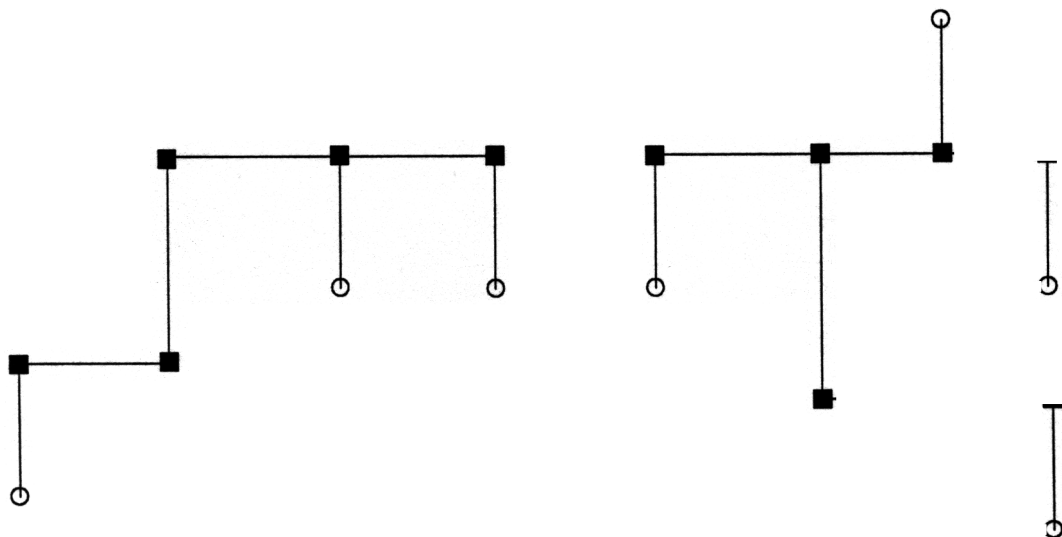


FIG. 4.19 TYPE 6 NET SAMPLES

#### 4.7.7 CATEGORY 7 FEATURE VECTOR

The feature vector for Type 7 nets follows:

$$F_7 : \{ X_1, X_2, X_3, X_4, X_5, X_6 \mid (\text{conditions}) \}$$

where the conditions are:

$$X_2 == X_1 - [(X_5 - 1) * 2];$$

$$X_3 \geq 1;$$

$$X_4 == 0;$$

$$X_5 > 1;$$

$$X_6 == 0;$$

Sample nets of Type 7 are shown in Fig. 4.20.

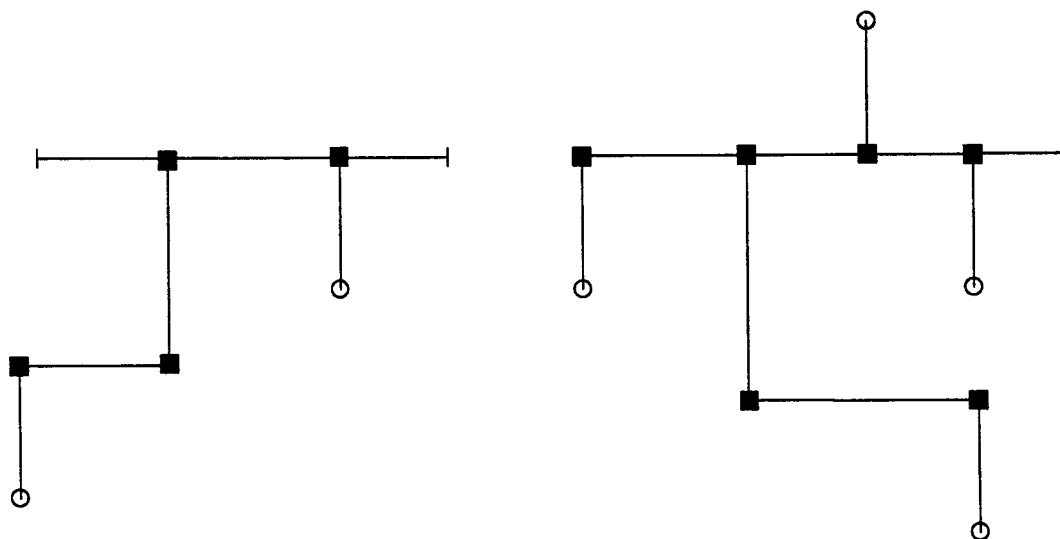


FIG. 4.20 TYPE 7 NET SAMPLES

#### 4.7.8 CATEGORY 8 FEATURE VECTOR

The feature vector for Type 8 nets follows:

$$F_8 : \{ X_1, X_2, X_3, X_4, X_5, X_6 \mid (\text{conditions}) \}$$

where the conditions are:

$$X_2 == X_1 - [(X_5 - 1) * 2] - X_4;$$

$$X_3 == 0;$$

$$X_4 \geq 1;$$

$$X_5 > 1;$$

$$X_6 == 0;$$

Sample nets of Type 8 are shown in Fig. 4.21.

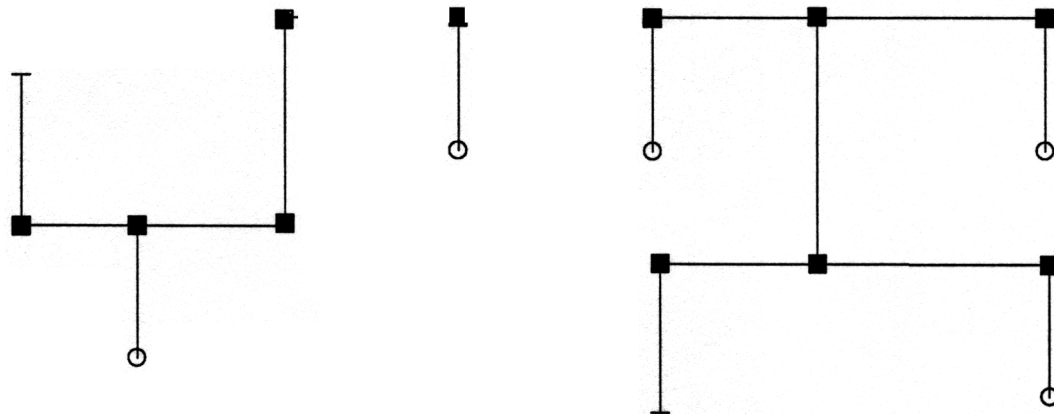


FIG. 4.21 TYPE 8 NET SAMPLES

#### 4.7.9 CATEGORY 9 FEATURE VECTOR

The feature vector for Type 9 nets follows:

$$F_9 : \{ X_1, X_2, X_3, X_4, X_5, X_6 \mid (\text{conditions}) \}$$

where the conditions are:

$$X_2 == X_1 - [(X_5 - 1) * 2] - X_4;$$

$$X_3 \geq 1;$$

$$X_4 \geq 1;$$

$$X_5 > 1;$$

$$X_6 == 0;$$

Sample nets of Type 9 are shown in Fig. 4.22.

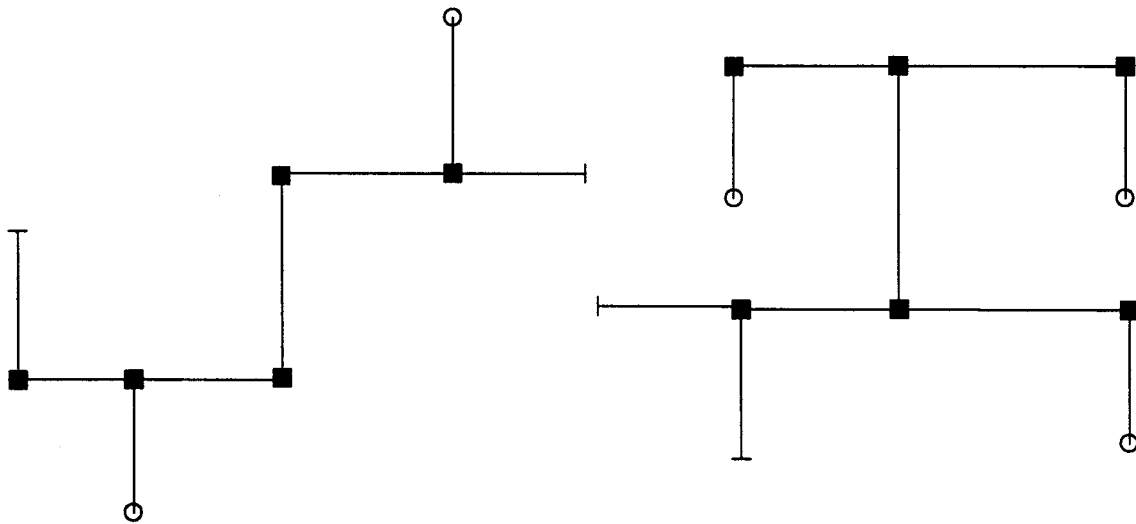


FIG. 4.22 TYPE 9 NET SAMPLES

#### 4.7.10 CATEGORY 10 FEATURE VECTOR

The feature vector for Type 10 nets follows:

$$F_{10} : \{ X1, X2, X3, X4, X5, X6 \mid (\text{conditions}) \}$$

where the conditions are:

$$X2 == X1 - [(X5 - 1) * 2] + X6;$$

$$X3 == 0;$$

$$X4 == 0;$$

$$X5 > 1;$$

$$X6 \geq 1;$$

Sample nets of Type 10 are shown in Fig. 4.23.

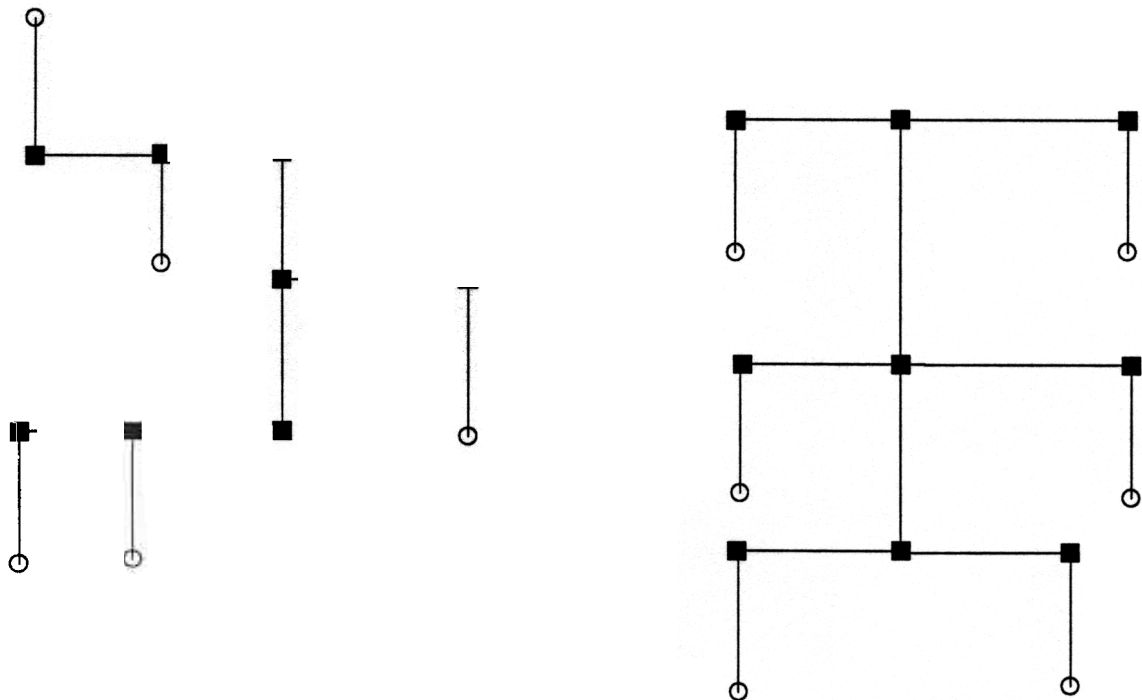


FIG. 4.23 TYPE 10 NET SAMPLES

#### 4.7.11 CATEGORY 11 FEATURE VECTOR

The feature vector for Type 11 nets follows:

$$F_{11} : \{ X1, X2, X3, X4, X5, X6 \mid (\text{conditions}) \}$$

where the conditions are:

$$X2 == X1 - [(X5 - 1) * 2] + X6;$$

$$X3 \geq 1;$$

$$X4 == 0;$$

$$X5 > 1;$$

$$X6 \geq 1;$$

Sample nets of Type 11 are shown in Fig. 4.24.

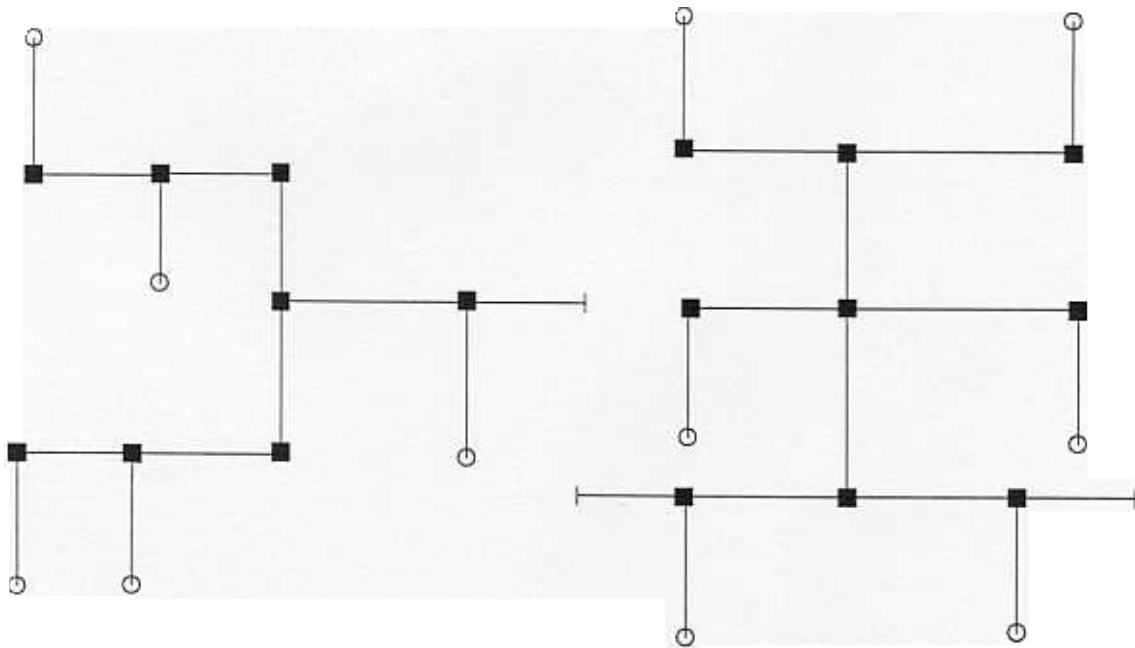


FIG. 4.24 TYPE 11 NET SAMPLES

#### 4.7.12 CATEGORY 12 FEATURE VECTOR

The feature vector for Type 12 nets follows:

$$F_{12} : \{ X1, X2, X3, X4, X5, X6 \mid (\text{conditions}) \}$$

where the conditions are:

$$\begin{aligned} X1 &== X2; \\ X2 &== X1 - [(X5 - 1) * 2] + X6 - X4; \\ X3 &== 0; \\ X4 &\geq 1; \\ X5 &> 1; \\ X6 &\geq 1; \end{aligned}$$

Sample nets of Type 12 are shown in Fig. 4.25.

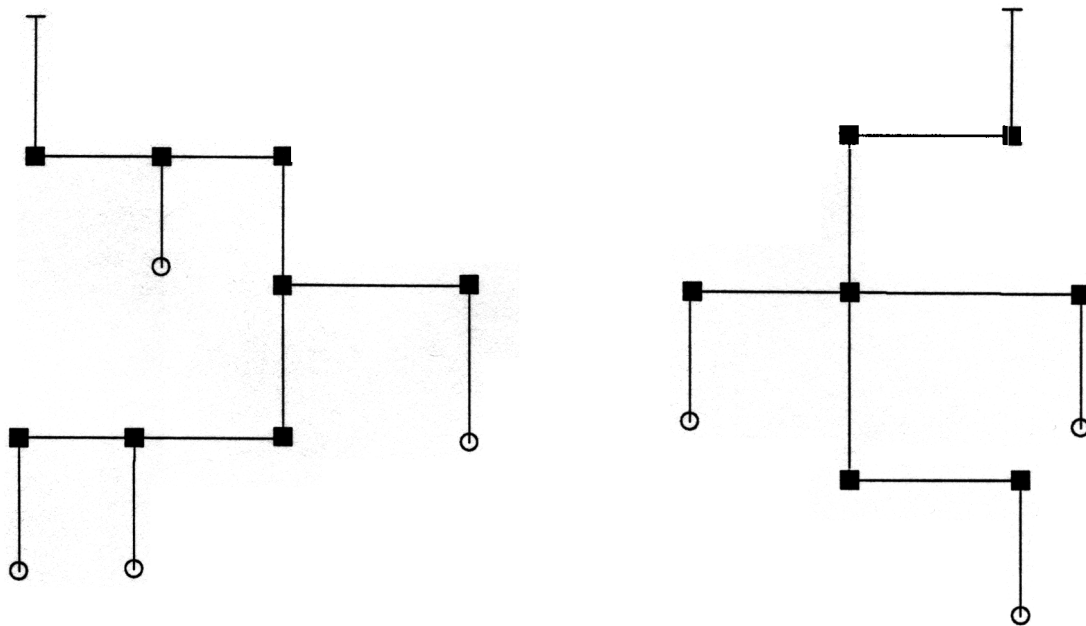


FIG. 4.25 TYPE 12 NET SAMPLES



### 4.7.13 CATEGORY 13 FEATURE VECTOR

The feature vector for Type 13 nets follows:

$$F_{13} : \{ X1, X2, X3, X4, X5, X6 | (\text{conditions}) \}$$

where the conditions are:

$$X2 == X1 - [(X5 - 1) * 2] + X6 - X4;$$

$$X3 \geq 1;$$

$$X4 \geq 1;$$

$$X5 > 1;$$

$$X6 \geq 1;$$

Sample nets of Type 13 are shown in Fig. 4.25.

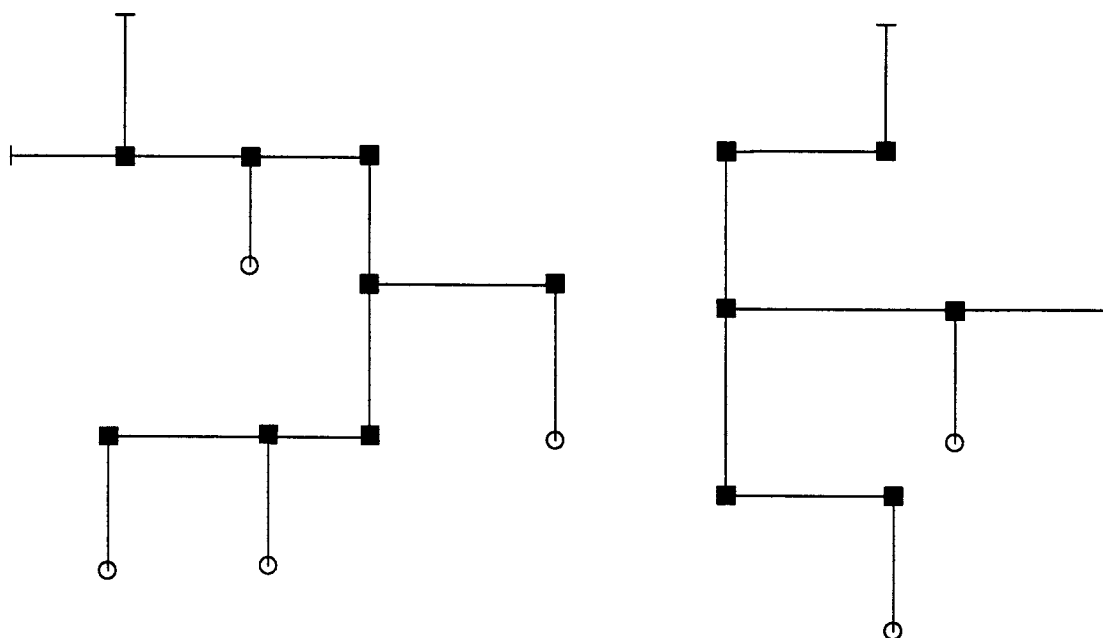


FIG. 4.26 TYPE 13 NET SAMPLES

## 4.8 SUMMARY

The idea to employ the computer vision concept of feature vectors was not immediate. Instead, it evolved gradually, as the first tentative steps toward determining the feasibility of the fat wire bifurcation problem. Applying them to the net recognition problem proved both reasonable and efficient. Typical vector components were discrete units rather than probabilities. This meant that the combinatorics normally associated with their use could be bypassed.

The categories or classes of nets grew from a static analysis of typical fat wire routing files. As additional topologies were identified, a recognition vector was developed to detect it. After a reasonable number of categories had been established, an underlying structure emerged. The first four categories of net types seemed to form a primitive group from which the more complicated, multiple backbone nets could be constructed. Finally, the true discriminators were identified, and the final taxonomy tree constructed.

A study of the steps involved in identifying the appropriate vector, based on assessing the relative conditions of each of the thirteen vectors, showed an interesting redundancy. By making the connection between the redundancies involved in vector checking and the structure of the taxonomy tree, an identification sieve was postulated. By structuring the vector condition tests so as to follow the tree decision branches, a complete vector identification could be completed in time proportional to the height of the tree. This was a major reduction from the time necessary to test all conditions of all vectors to arrive at a conclusion.

With linear time net recognition a reality, only two things still remained to be done. First, prove that the feature vectors accurately partitioned nets into disjoint sets, and then, derive an algorithm that would split each recognized net in linear time. Chapters five and six deal with these topics.

---

---

## **CHAPTER 5**

### **Managing Differential Signal Placement**

### **FINITE STATE MACHINE THEORY & REGULAR EXPRESSIONS**

---

#### **5.0 GENERAL THOUGHTS**

The empirical experience gained from generating the feature vectors of chapter four, and the apparent relation to the net taxonomy that developed was encouraging. However, without a theoretical foundation, there was no proof of correctness to justify the class equivalence of nets, nor the feature vector recognition mechanism. By drawing upon Finite State Machine (FSM) theory, it was possible to construct recognizers that served two functions. First, a fat wire net could be mapped by component into a FSM form. This form guaranteed that the fat wire net could be successfully bifurcated and the proper polarities propagated to all terminals of the net, with necessary freedom available to apply inversions. Second, if the tokens from a net were used as inputs to a FSM recognizer, its finishing in an acceptance state effectively categorized the fat wire net by type. Since successful bifurcation could be assured, and the net recognized by type, specialized bifurcation routines could be invoked to handle the splitting operation.

Indirectly, this provided a means to partition the splitting process, and thereby reduce the complexity of the software verification problem.

This chapter begins by looking at the original fat wire net to FSM mapping operation, which validated the equivalence among topologies in a taxonomy class. From that point, FSM recognizers are constructed for each node of the taxonomy tree. With this set of recognizers, any bifurcatable net can be identified. This finding validates the hypothesis that the taxonomy tree represents a basis set of net types from which all bifurcatable nets can be constructed. Finally, and most importantly, a link was identified which established the theoretical justification of the feature vectors developed in chapter four.

## **5.1 FINITE STATE MACHINE MAPPINGS**

As the taxonomy tree presented in chapter four was evolving, questions arose about the correctness of placing nets with apparently different topologies in the same node for bifurcation. Intuition would argue that differences in topologies would have formed the primary basis for partitioning the categories and building the taxonomy tree. Yet, as the feature vectors were generated in the earlier chapter, there appeared to be justification for categorizing nets on an underlying structure, not on obvious topology. Uncovering this underlying characteristic was the problem.

It was discovered that FSM theory could be brought to bear on the problem. By beginning at one port location of a net and using it as the start state, a walk along the net could be conducted. As each component of the net is encountered, it serves as the input token for that particular state. To successfully migrate from state to state, the component under consideration has to permit the proper propagation of polarity. If at any juncture, the proper polarity can not be extended to the next state, the net cannot not be properly mapped. This inability to complete the mapping served to identify nets as not bifurcatable.

Conversely, any net that successfully undergoes the mapping operation is bifurcatable, and the category of net to which it belongs is discernible. By identifying nets which are not bifurcatable at the outset, computational time can be conserved. Whereas, advocates of a generalized bifurcation algorithm would have to wait until the search space was exhausted before it was known that a net was not bifurcatable.

The first step in applying the FSM strategy was to establish the algorithms necessary for net to FSM mapping. Preliminary algorithms were developed to deal with nets that mapped into each of the nodes of the taxonomy tree. An example of net to FSM mapping is given below. The SLE net in Fig. 5.1 will be used. One of the ports of the net is selected. It becomes the start state  $S_0$ , in Fig. 5.2. The allowable input while in the start state is a vertical segment to a via (VSV).

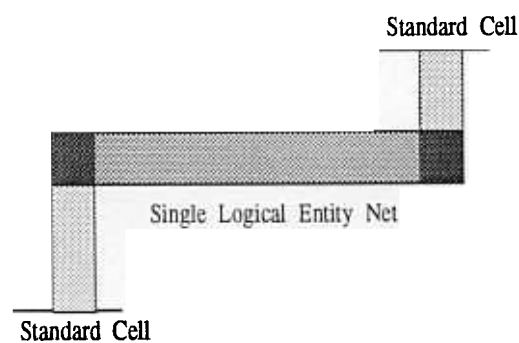


FIG. 5.1 SLE NET FOR FSM MAPPING

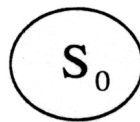


FIG. 5.2 START STATE MAPPING

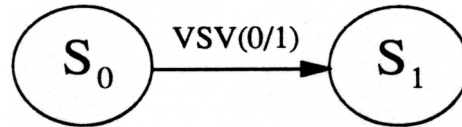


FIG. 5.3 LINK TO STATE S1

This is represented as an arc to state  $S_1$ , Fig. 5.3. The polarity propagation is depicted as an output of this transition and is shown in parenthesis. The allowable input for state  $S_1$  is a horizontal segment to a via (HSV), which possesses one degree of freedom. This transition is represented as an arc labeled HSV going to state  $S_2$ , with the polarity propagated as an output, again in parenthesis in Fig. 5.4. For State  $S_2$  the allowed inputs

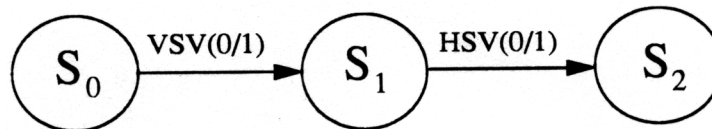


FIG. 5.4 TRANSITION TO S2

are a vertical segment to an instance (VSI) or a HSV. If it is a VSI then show it as an arc labeled VSI to an acceptance state  $A_1$ . Since state  $S_2$  has degree of freedom available to it, either (0/1) or (1/0) polarity can be propagated to the instance. This implies that the correct polarity can be propagated from start state  $S_0$  to acceptance state  $A_1$ . In this example, the specific FSM in Fig. 5.5 is in reduced form with no redundant states.

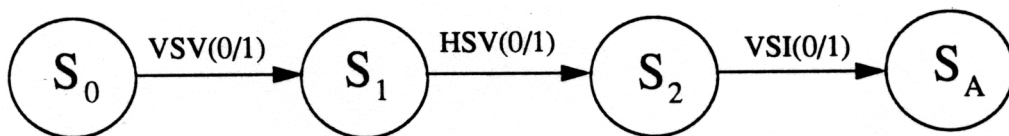


FIG. 5.5 FINAL STATE MACHINE

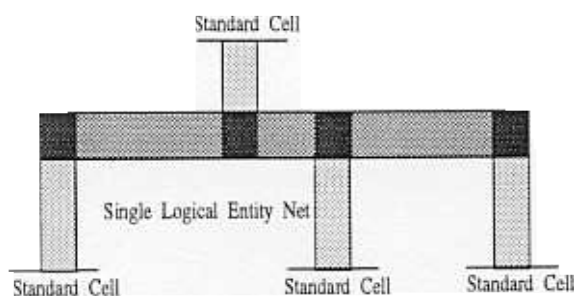


FIG. 5.6 SLE MAPPING EXAMPLE

A more representative net is provided as an example in Fig. 5.6. and the FSM mapping produces the machine shown in Fig. 5.7. If all net ports, other than the one selected as the start state  $S_0$ , map successfully into acceptance states, all acceptance states  $A_1$ ,  $A_2$ , and  $A_3$  can be grouped together as in Fig. 5.8. The subscripts on the states  $S_2$ ,  $S_3$  and  $S_4$  serve to show them as distinct during the original mapping operation.

Upon closer analysis, each can be seen to be the state  $S_2$  as defined by the allowable inputs and the corresponding transition function. Thus, the FSM can be re drawn showing each of the three states re labeled as  $S_2$ , and combining the acceptance states. This is shown in Fig. 5.9.

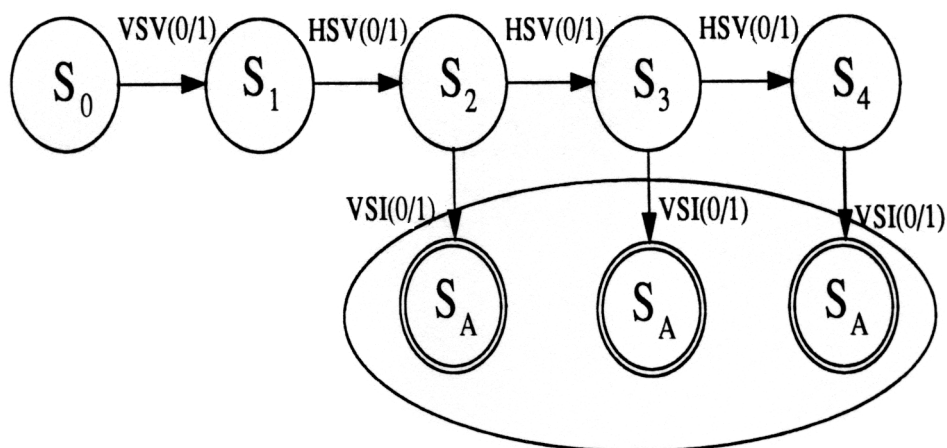


FIG. 5.7 PRELIMINARY SLE NET MAPPING

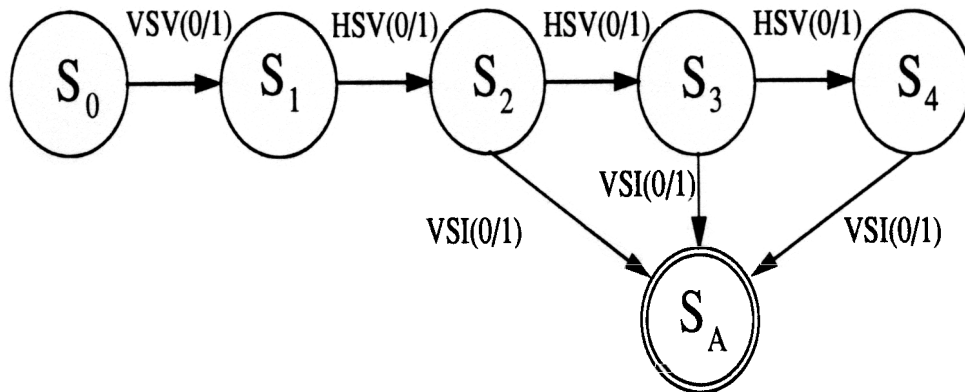


FIG. 5.8 FSM WITH ACCEPTANCE STATES CONSOLIDATED

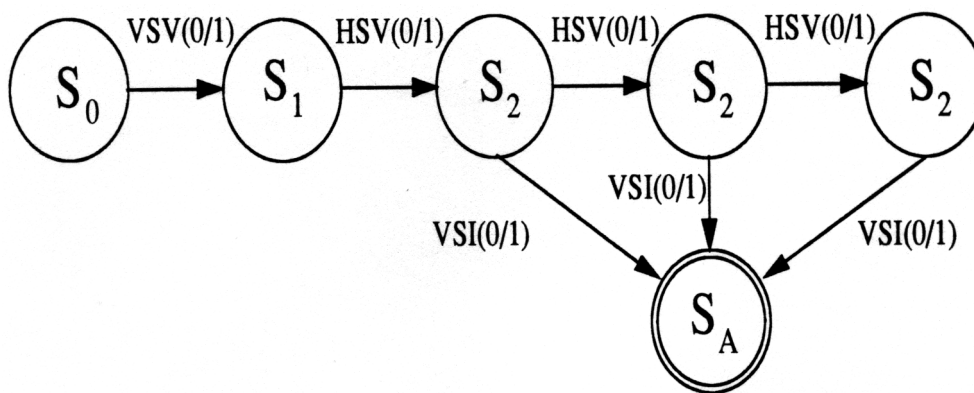


FIG. 5.9 PRECURSOR STATES LABELED AS S2

Finally, by eliminating redundant states and re drawing, the FSM shown in Fig. 5.10 is generated.

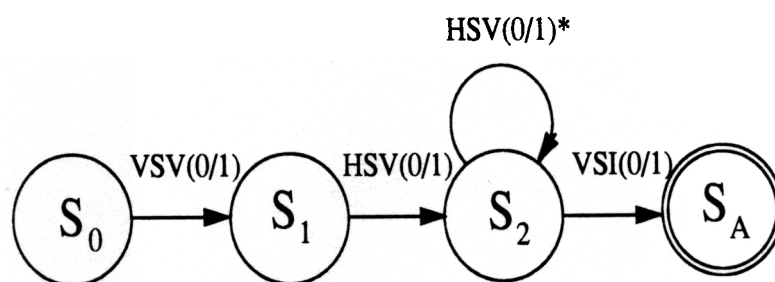


FIG. 5.10 REDUCED FSM



From this development one can observe that the finite state machine derived in the first example is really a subset of the second, since the HSV input is an allowed input for state  $S_2$ . As it turns out, the second version is the generalized finite state machine representing one of the taxonomy nodes. By means of a similar analysis other FSMs corresponding to tree nodes can be derived. This theoretical analysis demonstrates that the bifurcation operation can be partitioned into a finite number of separate routines, each designed to handle a particular primitive that corresponds to a node in the taxonomy tree. This conclusion appeared to be identical to the result obtained when feature vectors were employed as the analysis tool. However, one problem still remains. There had to be a way to prove that the categorization produced by the feature vectors was identical to the one derived by the FSM approach. This is important for two reasons.

From a theoretical perspective, the categorization effected by the FSM technique was solid. Nets could be partitioned by type and at the same time one could be assured of proper bifurcation and polarity propagation. Yet implementation of such an approach in actual code would be extremely complex. Linking this theoretical approach to the feature vector technique would allow a rapid and straightforward implementation.

Second, empirically derived feature vectors lack the definitiveness of the FSM theory. It could be argued intuitively that the vectors were all disjoint, and produced the proper categorization. But in the VLSI CAD regime, where errors result in meaningless circuits, and the waste of enormous amounts of fabrication capital, a more rigorous proof of correctness was needed.

The formal link between the two approaches is presented in section 5.4. But first, a flaw in the FSM mapping theory must be corrected.

## 5.2 REVISED FSM MAPPING

The preliminary FSM mapping algorithm presented in section 5.1, was originally developed around the primitive net categories (1 - 5) in the taxonomy tree. Since the multiple backbone configurations are extensions of the primitives, the extendibility of the mapping procedure seemed obvious. However, as refinements were made, and the final algorithms and state machines reviewed for inclusion in the dissertation, a flaw in the mapping theory was uncovered.

For single backbone nets, a “walk” along the net was straight forward, with a single input at any given node. This matched the transition criteria for finite state machines. Beginning in a state, review an input, and based on that input execute a transition to another state. (This “other” state could in fact be the same state.) The difficulty arises when fork junctions occur, and both avenues must be explored. This is like saying that there are two valid inputs at a state and that both transitions are correct and somehow must be explored simultaneously. Such assumptions would seriously stretch the FSM theory and make the findings produced by such assumptions suspect.

In order to continue to use the finite state machine theory in its pure form, the decision was reached to postulate a more powerful automaton to serve as the tokenizer for the net under examination. Such an automaton would effectively partition multiple backbone nets into their constituent single backbone components, keeping track of the partition that was introduced. It would follow the cut line concept put forth in chapter three. In so doing, the multiple backbone identifiers were then constructed as unions of single backbone recognizers. The revised FSM recognizers are presented in section 5.3.

### 5.3 NET RECOGNIZERS

Employing the modified mapping technique, the recognizers for the thirteen categories in the net taxonomy tree were constructed. They are presented on the following pages. The representative nets provided in Figs. 4.14 - 4.26 can be consulted to visualize the net topology recognized by each machine.

#### 5.3.1 TYPE 1 RECOGNIZER

The following state machine will recognize and accept a Type 1 Fat Wire Net:

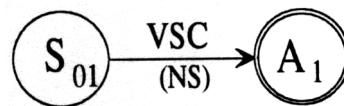


FIG. 5.11 TYPE 1 NET RECOGNIZER

#### 5.3.2 TYPE 2 RECOGNIZER

The following state machine will recognize and accept a Type 2 Fat Wire Net:

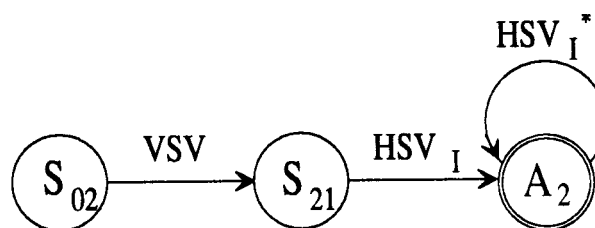


FIG. 5.12 TYPE 2 NET RECOGNIZER

### 5.3.3 TYPE 3 RECOGNIZER

The following state machine will recognize and accept a Type 3 Fat Wire Net:

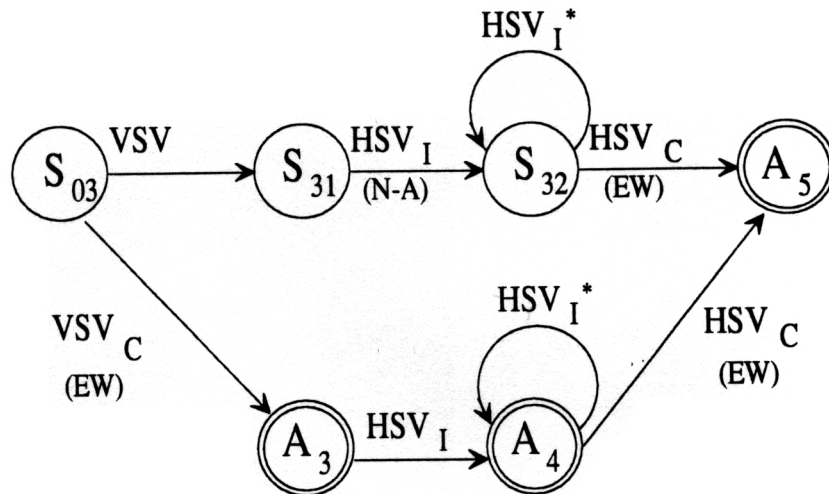


FIG. 5.13 TYPE 3 NET RECOGNIZER

### 5.3.4 TYPE 4 RECOGNIZER

The following state machine will recognize and accept a Type 4 Fat Wire Net:

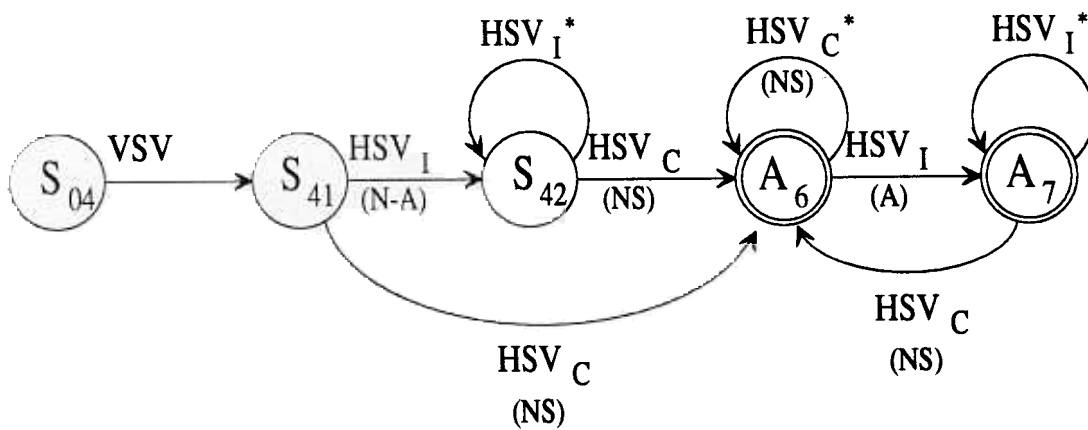


FIG. 5.14 TYPE 4 NET RECOGNIZER

### 5.3.5 TYPE 5 RECOGNIZER

The following state machine will recognize and accept a Type 5 Fat Wire Net:

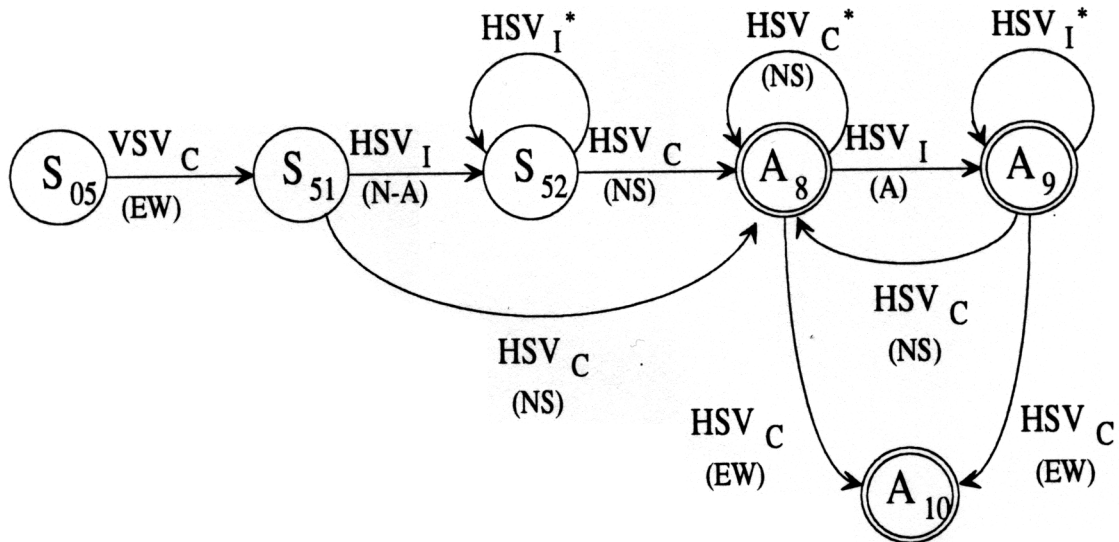


FIG. 5.15 TYPE 5 NET RECOGNIZER

### 5.3.6 TYPE 6 RECOGNIZER

The following state machine will recognize and accept a Type 6 Fat Wire Net:

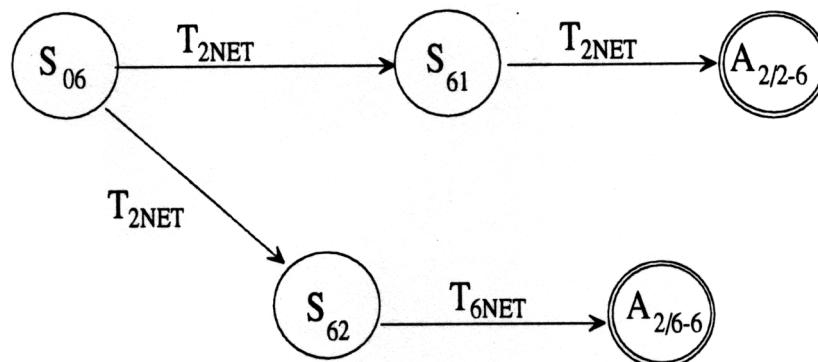


FIG. 5.16 TYPE 6 NET RECOGNIZER

### 5.3.7 TYPE 7 RECOGNIZER

The following state machine will recognize and accept a Type 7 Fat Wire Net:

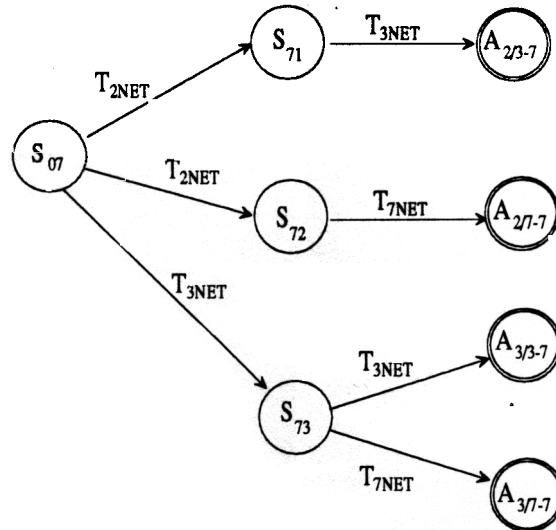


FIG. 5.17 TYPE 7 NET RECOGNIZER

### 5.3.8 TYPE 8 RECOGNIZER

The following state machine will recognize and accept a Type 8 Fat Wire Net:

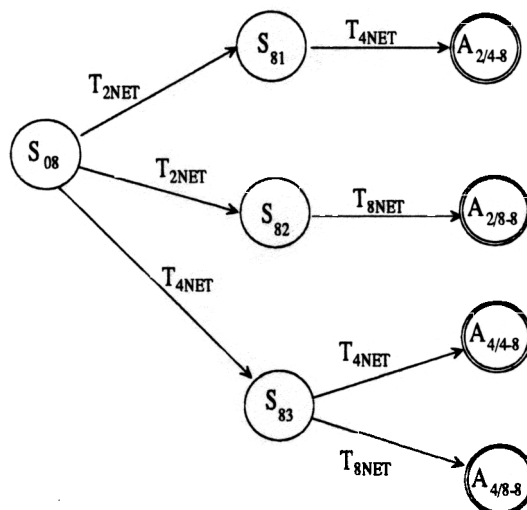


FIG. 5.18 TYPE 8 NET RECOGNIZER

### 5.3.9 TYPE 9 RECOGNIZER

The following state machine will recognize and accept a Type 9 Fat Wire Net:

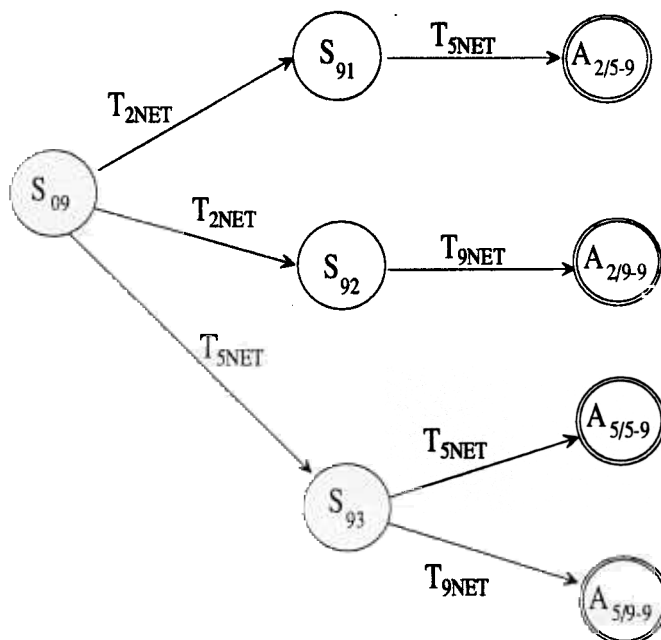


FIG. 5.19 TYPE 9 NET RECOGNIZER

### 5.3.10 TYPE 10 RECOGNIZER

The following state machine will recognize and accept a Type 10 Fat Wire Net:

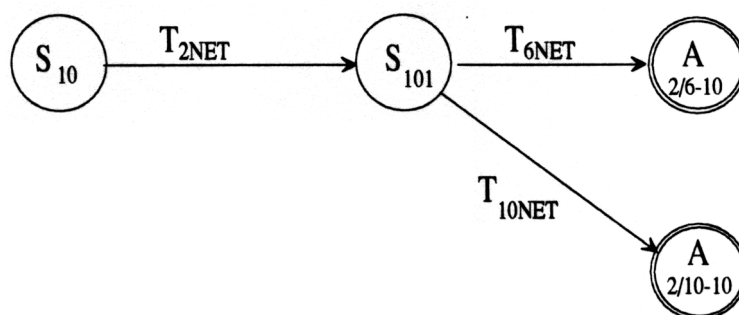


FIG. 5.20 TYPE 10 NET RECOGNIZER

### 5.3.11 TYPE 11 RECOGNIZER

The following state machine will recognize and accept a Type 11 Fat Wire Net:

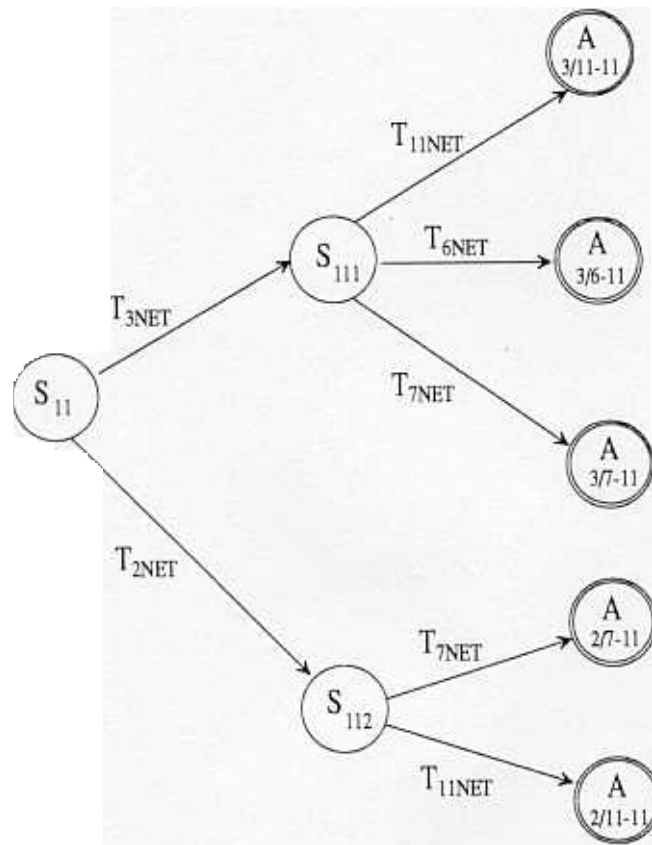


FIG. 5.21 TYPE 11 NET RECOGNIZER



### 5.3.12 TYPE 12 RECOGNIZER

The following state machine will recognize and accept a Type 12 Fat Wire Net:

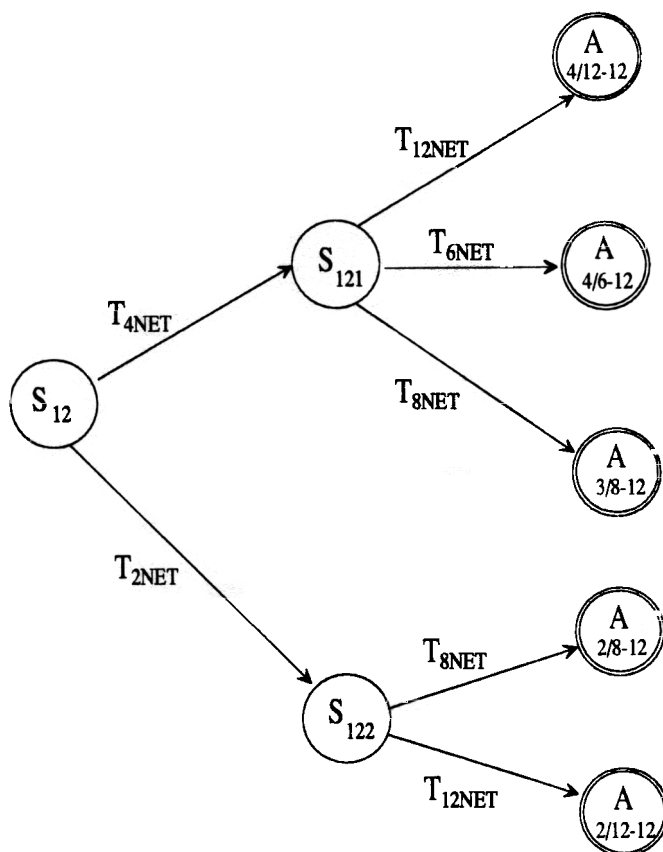


FIG. 5.22 TYPE 12 NET RECOGNIZER

### 5.3.13 TYPE 13 RECOGNIZER

The following state machine will recognize and accept a Type 13 Fat Wire Net:

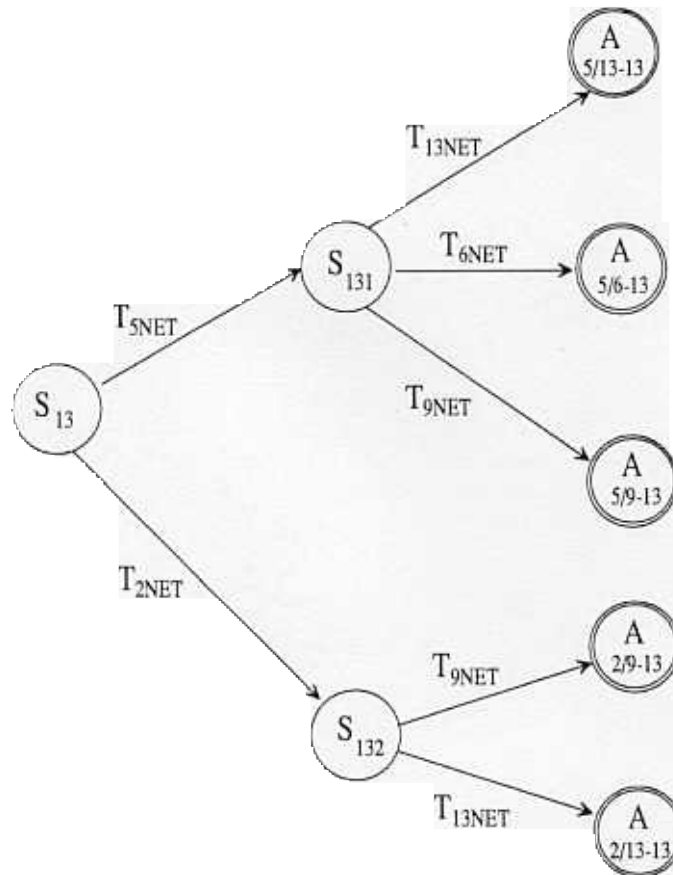


FIG. 5.23 TYPE 13 NET RECOGNIZER

## 5.4 REGULAR EXPRESSIONS - (THE FORMAL LINK)

As was pointed out in section 5.1, the FSM Theory provides an elegant analytical tool for examining fat wire nets, and a theoretical proof of correctness for proper polarity propagation. In one step, a fat wire net can be certified to be bifurcatable and at the same time categorized into one of the basic taxonomy classes. The drawback to employing the theory is the difficulty encountered at implementation time. Translating these concepts into

working code is an extremely difficult undertaking. The challenge was to link this formalism with the feature vector concept of chapter four. If this could be accomplished, then the feature vectors could be used to achieve the same goals: (1) certification of bifurcatability, and (2) categorization; but both in easily understood code that runs in linear time.

After working with the state machines for several months during their revision process, the link between the two techniques which had evaded me for so long finally revealed itself. If one takes the arc labels from the finite state machines, a regular expression can be generated. This regular expression is equivalent to the FSM in that it represents the language,  $L$ , of strings recognized by the FSM. By labeling the links of the FSMs with their respective notation, the feature vector equivalents were immediately obvious. It was the regular expressions, derived from the recognizers, that provide the theoretical link to the set of feature vectors. This finding was distributed to all committee members as a research update in March, 1993.

Nets decomposed into tokens can be used as inputs to FSM recognizers. Using limited memory, arbitrarily long inputs (corresponding to complex nets) can be identified in a finite amount of space.

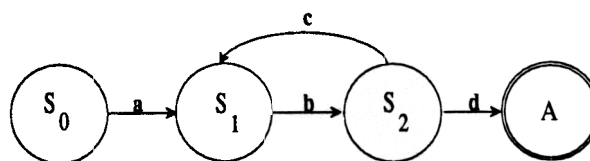


FIG. 5.24 SAMPLE FSM TO GENERATE REGULAR EXPRESSION

A representative FSM is depicted in Fig. 5.24. The corresponding regular expression is shown in equation 5-1.

$$\text{Reg\_Expression} = a b (c b)^n d \quad (5-1)$$

In this form, the strict mathematical relations among the components is established. For equation 5-1, the constraints are:

$$\{ a == 1 \}, \{ d == 1 \}, \{ b \geq 1 \}, \{ \# c == \# b - 1 \}$$

Using the FSM recognizers constructed previously, and re labeling their arcs using graphic symbology, the feature vectors for all categories could be generated directly. This operation is demonstrated in the following figures.

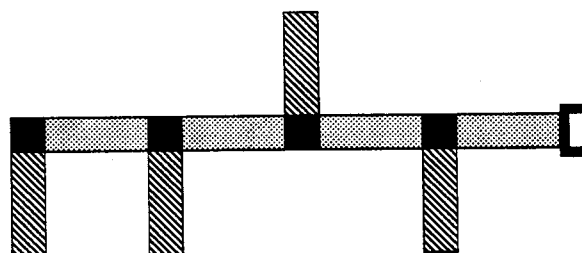


FIG. 5.25 (A) SLE NET

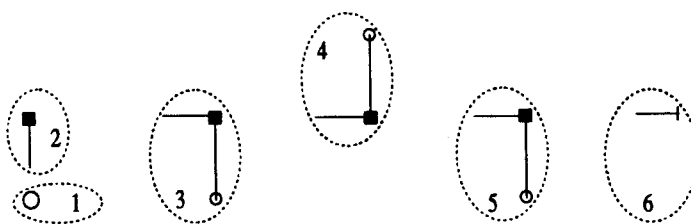


FIG. 5.26 NET SPLIT INTO GRAPHICAL INPUT TOKENS

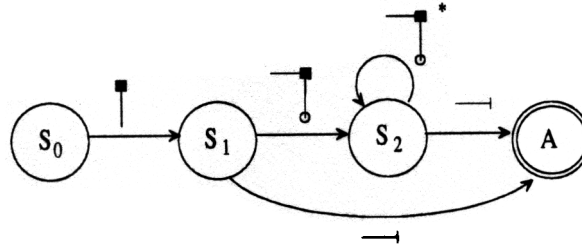


FIG. 5.27 FUNCTIONING FSM RECOGNIZER

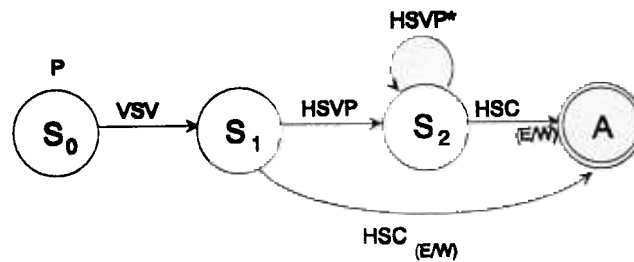


FIG. 5.28 RECOGNIZER WITH FORMAL LABELS

When the regular expression is written using the formal FSM labels, it appears as-

$$P(vseg)V\{(hseg)VP\}^*$$

Discounting the vseg(vertical segment) and hseg(horizontal segment) entries, the expression can be rewritten as-

$$PV\{VP\}^*$$

From this expression, the feature vector conditions for this type net are written directly as-  
 $\{ Vias \geq 1, Ports \geq 1, E/W Conn. == 0, N/S Conn. == 0, Layers == 1, \# Ports == \# Vias \}$

The finite state machine (FSM) approach, makes it possible to draw on regular grammar theorems. From Automata theory we know that for distinct finite state machines, the corresponding regular expressions will be distinct [McNa84]. Ideally, this theorem could be used to insure a disjoint set of feature vectors. But, by dropping the FSM structure, and only focusing on the relation among components of the regular expression, the converse does not follow. Since the set of feature vectors was limited (thirteen), they

can be shown to be distinct by proof through enumeration. A full listing of the feature vectors appeared in chapter four.

## 5.5 SUMMARY

As the fat net recognition problem progressed, a means of proving equivalence among certain topologies was needed. Additionally, a technique was sought that would prove that a given category of net in the taxonomy tree could be correctly bifurcated. Finite state machine theory provided the ideal tool to satisfy these requirements. If the net to FSM mapping was carried to successful completion, then it showed that the net was bifurcatable. Having constructed FSM recognizers, and using the net components as input tokens, an given fat wire net could be both recognized by type and certified as bifurcatable. Any net that was not recognized, by definition, could not be guaranteed to be bifurcated by the system. Some characteristic of its topology demanded a larger degree of freedom than what was provided by the available via junctions.

Using arc labels from the recognizers, regular expressions for each category of net in the taxonomy tree were constructed. These provided the theoretical association to the feature vectors presented in chapter four. Showing this equivalence was of paramount importance, since feature vector methodology was both readily implementable, and enjoy a linear time complexity.

With the feature vector techniques of chapter four supported by the FSM theory, the bifurcation problem posed by Stage III of the router architecture can now be effectively handled.

---