Chapter 5

Structural Analysis of Mathematical Expressions

5.1 Introduction

In this chapter, we describe a structural analysis method for the recognition of on-line handwritten mathematical expressions based on a *minimum spanning tree* (*MST*) construction and symbol dominance. Matsakis [61] proposed a MST construction as a "starting point" for the structural analysis of mathematical expressions. He considers the strokes as the nodes of a totally connected weighted graph. His method groups strokes into symbols, obtaining a tree which connects symbols in the expression and describes in approximation the final structure of the expression, see Fig. 5.1(a)-(b). Unfortunately, this does not occur when dealing with more complex expressions as the one shown in Fig. 5.1(c)-(d).

We also consider the recognized symbols as the nodes of a totally connected weighted graph. Our method handles some layout irregularities frequently found in on-line handwritten formula-recognition systems, like irregular horizontal layouts and symbol overlapping. It also handles arguments of operators with non-standard layouts as well as tabular arrangements, like matrices.

In our method, the crucial step in the MST construction will be the weight calculation of edges. We use three main constructions. The first one is a "preprocessing" step during the structural analysis. It helps to handle irregular horizontal layouts by locating fraction bars in the expressions. The second MST construction helps to group symbols in clusters. They are associated recursively to spatial regions which define mathematical relations. The last construction is used to locate rows in matrices.

In this chapter we define algorithms, functions, and relations for symbols and

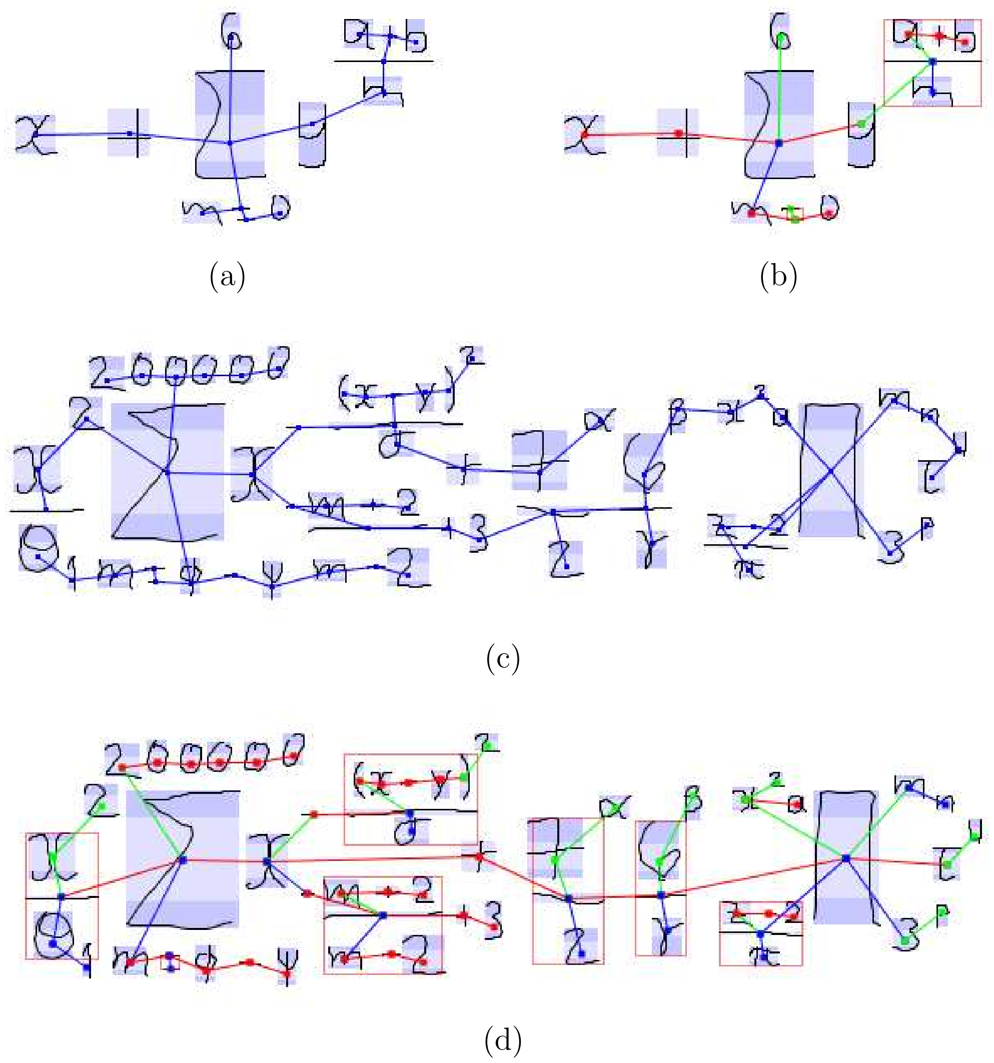


Figure 5.1: *Figures (a)-(c): the minimum spanning tree. Figures (b)-(d): the desired relation tree.*

other data structures. Because most of them are easy to implement, we do not give their implementations, pseudo-code, or return values, all of them should be easily derived through the function's name. It should also be clear from the context that some functions can modify their arguments, as done in the language C, when passing variables by reference. Section 5.2 introduces the concepts and data structures we need for the rest of the chapter. In Sect. 5.3 we describe the construction of the minimum spanning tree based on symbol dominance. Section 5.4 concludes with a discussion about our method.

5.2 Structural Analysis

5.2.1 Symbol Regions and Symbol Attributes

Relations and operator dominance in mathematical notation are defined explicitly or implicitly by the position and relative size of symbols in an expression. The spatial regions *above-left*, *above*, *superscript*, *right*, *subscript*, *below*, *below-left* and *subexpression* are used to determine such relations. For example, the operands (numerator and denominator) of the horizontal bar (fraction operator) are expected to lie in the regions *above* and *below* of the horizontal bar. See Fig. 5.2.

By comparing symbol attributes, we can test whether or not a symbol belongs to a determined spatial region. Given a symbol s, we consider its *label* and its *bounding box* as the *basic attributes.* The label is obtained by means of a classifier, as described in the previous chapter. The bounding box is defined by the minimum x and y coordinates (x_s, y_s) of all points in the symbol, its height H_s, and weight W_s. Once raw symbols are endowed with attributes, we can collect them in *ordered attributed lists.* The order of a symbol in a list is determined by its leftmost x coordinate, i.e. if the list L is formed by the symbols $(s_1, \ldots, s_k)$, it means that $x_{s_i} \leq x_{s_j}$ for $i < j$. In this chapter, when we refer to a list we are talking about an ordered list. The attributes of a list L are its label and its bounding-box attributes (x_L, y_L), H_L and W_L. The label of a symbol list is obtained during the structural analysis process by the spatial and geometrical relations between symbols.

From the basic attributes, we derive the *superscript threshold* and the *subscript threshold.* They are numeric attributes used to delimit the regions around symbols. We also derive the *centroid.* It is a point attribute which determines the symbol's location in regions. To determine these symbol attributes, we classify the symbol as *ascendent*, *descendent*, or *central*, as shown in Table 5.1. The reason for doing so becomes clear if we observe the layout differences in the subindex relation of the central symbol x_* and the descendent symbol y_* (see Fig.5.2). The way to calculate attributes for a symbol s is given in Table 5.2. After obtaining symbol attributes we can determine which region a symbol lies in. For example, given the symbols s and a, we can define a boolean function to determine whether a lies in the above region of s, and other regions in a similar way, as follows:

```
liesInAboveRegion($s, a$)

    1. Return getMinX($s$) $\leq$ getCentroidX($a$) $\leq$ getMaxX($s$) &&
```

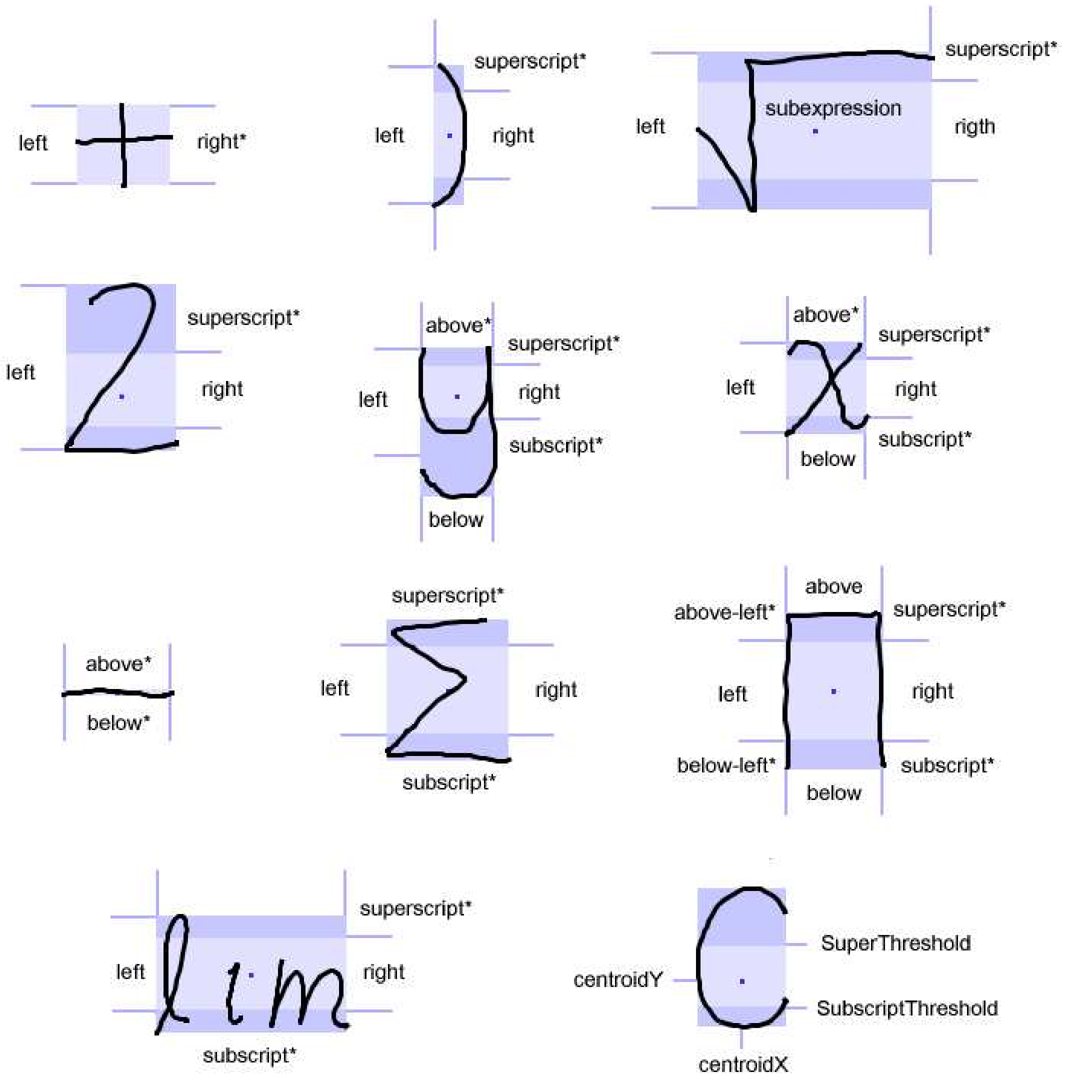


Figure 5.2: *Regions, thresholds, and centroids of different symbol types. From left to right: non-scripted, horizontal bar, square root, scripted, sum-like, and product operators. The regions marked with an asterisk determine the range of different symbol types.*

Table 5.1: *The symbols used in our system.*

	central	ascendent	descendent
non-scripted	$+ - * / (\infty \rightarrow$		
superscripted	e π $\sqrt{}$ log sin cos tan	0 1 2 3 4 5 6 7 8 9	
scripted	a c x z)	b d ∂ Δ ∇	g y
sum-like	$\sum \int \prod$		
lim-like	lim max min		

Table 5.2: *Attributes for different symbol types.*

	super threshold	sub threshold	centroid
ascendent	$y_s + 0.8H_s$	$y_s + 0.2H_s$	$(x + 0.5W_s, y + 0.33H_s)$
descendent	$y_s + 0.9H_s$	$y_s + 0.6H_s$	$(x + 0.5W_s, y + 0.66H_s)$
central	$y_s + 0.8H_s$	$y_s + 0.2H_s$	$(x + 0.5W_s, y_s + 0.5H_s)$

$$\texttt{getSuperThreshold}(s) \leq \texttt{getCentroidY}(a).$$

5.2.2 Symbol Dominance

The *range* of a symbol is the expected location area of its arguments, see Fig. 5.2. Chang [17] defines dominance as follows. A symbol s *dominates* a symbol a if a lies in the range of s and s does not lie in the range of a. We say that symbols *dominate* their arguments. Arguments have lower *precedence* than the dominant symbol.

We define `dominates`(s, a) as a boolean relation which depends on the set of *operator classes*

$$T = \{-, \sqrt{}, \text{scripted}, \text{superscripted}, \text{non-scripted}, \text{sum-like}\},$$

the spatial regions, and symbol attributes of s and a. The symbol '$-$' represents the horizontal bar and '$\sqrt{}$' the square root. If `dominates`(s, a) is true, it means that s dominates a. Observe that we added some extra conditions to the definition of

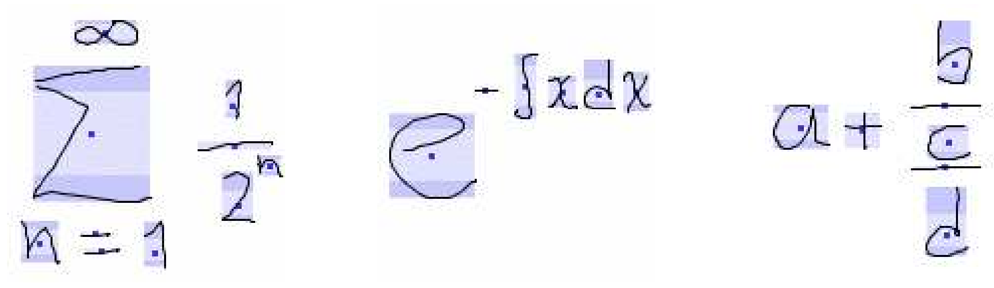

(a) (b) (c)

Figure 5.3: *Examples of expressions where (a) dominance is determined by range, (b) dominance is determined by considering symbol sizes and (c) dominance between fraction lines is hard to determine.*

Chang, namely comparison of symbol sizes and attributes, to determine dominance and to resolve ambiguity.

To clarify the concept of dominance, let us give a few examples. Consider the sum symbol in Fig. 5.3(a). It dominates the symbol '∞', because the later lies in the range (superscript region) of the first and we do not expect any symbol lying in any of the regions of '∞'. By analogy, the constant 'e' in Fig. 5.3(b) dominates the symbols '$-$' and '$\int$'. The horizontal bar lies in the superscript region of 'e', but the latter does not lie above or below the symbol '$-$'. Observe that 'e' lies in the range of the integral, but the dominance in this case is resolved by comparing their sizes. Figure 5.3(c) also shows a case where symbol dominance is not clear. We can not determine which one of the fraction lines dominates the other, because both of them lie in the range of the other and have the same size. We can avoid the confusion here by taking as the dominant fraction bar the one with the greater centroid's y-coordinate.

As we can see in these examples, dominance can be established by convention and can vary from one author to another. Different definitions of dominance define different *dialects* of mathematical notation.

5.2.3 Baseline Representation of Expressions

We describe mathematical notation as a hierarchical structure of nested baselines [103]. A *baseline* is a list which represents a horizontal arrangement of symbols in the expression. Each symbol has links to other baselines, which satisfy the spatial relations mentioned in Sect. 5.2.2, relative to it. The *dominant baseline* of an expression is the baseline which is not linked by any symbol. For example, the expression $x_{ij} * y + \frac{a+b}{c}$ is determined by the baselines $(x, *, y, +, -)$, (i, j), $(a, +, b)$ and (c). The last two baselines satisfy the relations *above* and *below* relative to the horizontal bar respectively. The dominant baseline of this expression is $(x, *, y, +, -)$.

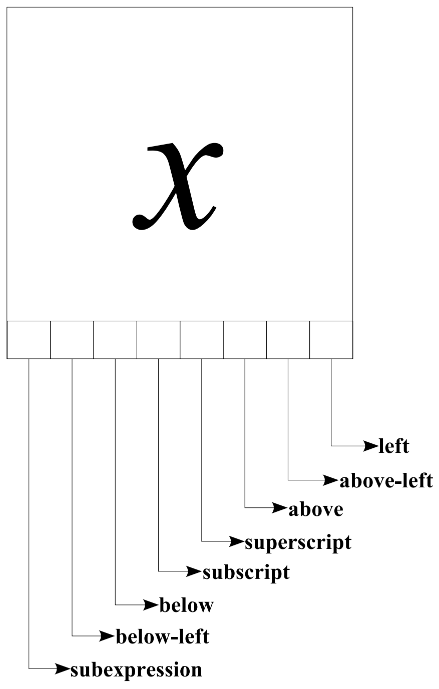
The data structure which represents the whole expression in the form described above is called *baseline tree*, see Fig. 5.4. This representation exploits the left-to-right reading of mathematical expressions. When reading an expression, one normally searches for the leftmost dominant symbol, then for the next leftmost dominant one, and so on until no more symbols are found. Given an ordered symbol list L, we can determine the leftmost dominant symbol in L through the function `getDominantSymbol`, which is defined as:

```
getDominantSymbol($L$)

    1. Let $n$ = length($L$).
    2. If $n == 1$, return $s_1$.
    3. If $s_n$ dominates $s_{n-1}$, remove $s_{n-1}$ from $L$, else remove $s_n$.
    4. Return getDominantSymbol($L$).
```

Observe that this function uses the order of symbols in L.

In this way, given a list L, we construct its dominant baseline Db through the function:

```
constructDominantBaseline($Db, L$)

    1. If $Db$ is empty, then set $Db$ = addSymbol($Db$, getDominantSymbol($L$)).
    2. Set $s$ = getLastSymbol($Db$).
    3. Construct a list $Hs$ = getRightNeighbors($s, L$) of symbols in $L$ which
       are right horizontal neighbors of $s$.
    4. If $Hs$ is empty, return.
    5. Find the dominant symbol of the horizontal neighbors,
       $sd$ = getDominantSymbol($Hs$).
    6. Set $Db$ = addSymbol($Db, sd$).
    7. Use recursion: constructDominantBaseline($Db, L$).
```



$$x_{ij} * y + \frac{a + b}{c}$$

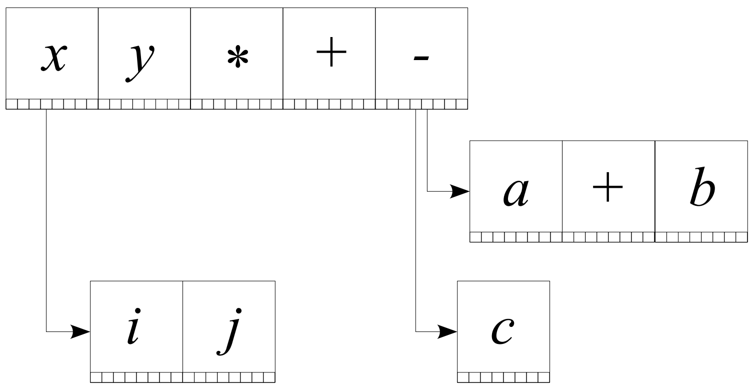


Figure 5.4: *Above: a node of the baseline tree stores the symbol's label and other attributes, it has also links which correspond to different spatial relations. Center: the original mathematical expression. Below: the representation of the expression as a baseline tree.*

We take special care in the definition of the function `getRightNeighbors` to handle irregular horizontal layouts.

Now we are ready to construct the baseline tree of the mathematical expression described by the ordered symbol list L by recursively finding dominant baselines. This is done by the function `constructBaselineTree`:

```
constructBaselineTree($L$)

    1. If $L$ is empty, return.
    2. Set $Db = \emptyset$.
    3. constructDominantBaseline($Db, L$).
    4. Update $Db$ by grouping together symbols which define operator and function names, like lim, sin, log, etc.
    5. constructDominanceMST($Db, L$).
    6. locateMatrices($Db, L$).
    7. For each symbol $s \in Db$, construct new symbol lists with its children obtained in the MST step, depending on which spatial relations they satisfy and assign these lists to the corresponding links. The identity of these lists corresponds to the spatial relation they satisfy.
    8. Set $L = Db$.
    9. For each symbol $s \in Db$, use recursion applying constructBaselineTree to each of its child lists obtained in step 7.
```

5.3 MST Construction and Symbol Dominance

The pseudo-code of the function `constructBaselineTree` outlines the procedure to construct the baseline tree. Steps 3, 5, and 6 are key steps to achive this purpose. Each of these steps requires an MST construction to handle the irregular horizontal layouts, to group symbols, and to locate rows in matrices respectively. The first MST construction is handled as a preprocessing step *prior to* the construction of the baseline tree. The last two are done at each recursion step of `constructBaselineTree`. The following sections explain how this is done.

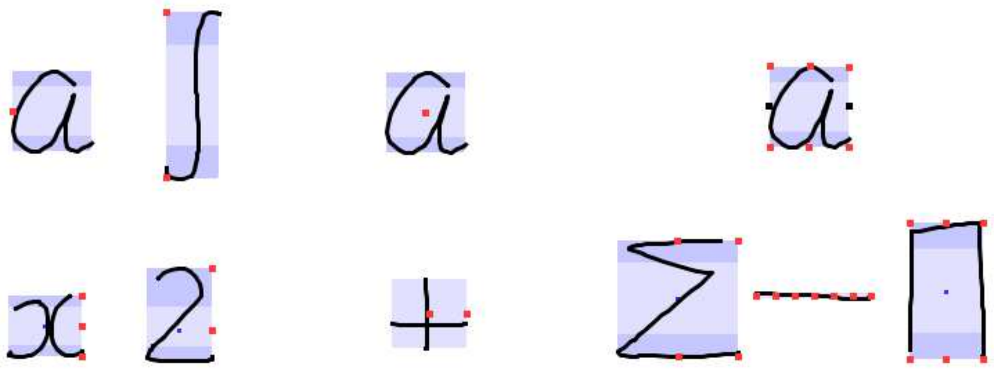


Figure 5.5: *Above: Attractor points of symbols not belonging to the MST. Below: Attractor points of symbols and operators belonging to the MST.*

MST Construction and Attractor Points

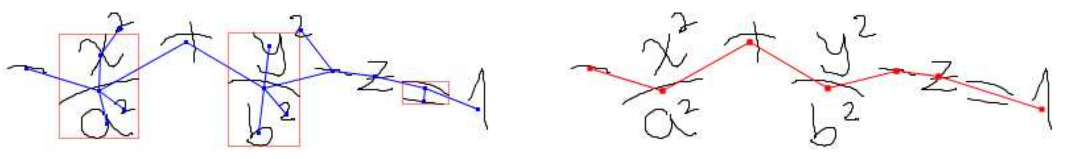
We consider the recognized symbols as the nodes of a totally connected weighted graph. The MST of such a graph is constructed as follows. Given the list L, we consider the symbols in L as the nodes of a totally connected weighted graph. Then, we use Prim's algorithm to construct its MST: a new edge (s_t, s_n) is added to the MST if its corresponding weight $w(s_t, s_n)$ is the minimum of all edges, where s_t belongs to the MST and s_n does not belong to the MST. This is the general MST algorithm for each of the constructions. They differ from each other in the way the weights are calculated.

The edge weight is the minimum distance between *attractor points* (AP) of symbols. They are located in the boundary of the symbol bounding box. The number of such points depends on the operator class. Figure 5.5 shows the attractor points corresponding to different symbol classes when the first a and the integral (sum-like operators and square root) are dominated by x (scripted) and 2 (superscripted), the second a is dominated by + (non-scripted), and when the third a is dominated by sum-like operators and the horizontal bar.

5.3.1 Construction of the Dominant Baseline

The Right Relation

The construction of the dominant baseline consists of finding symbols which describe horizontal arrangements in the expressions. The key is to have a robust definition of the relation "right horizontal".


(a)

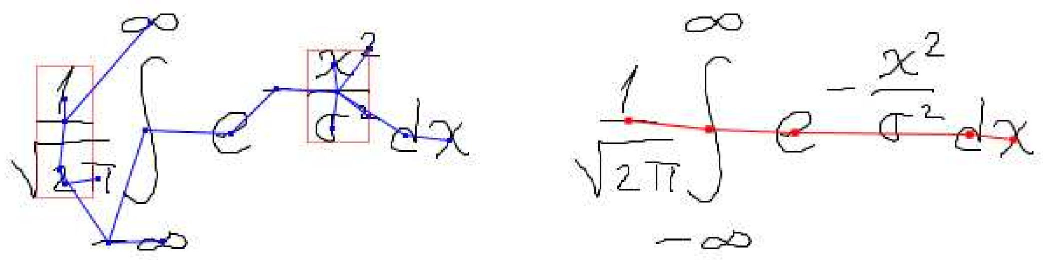

(b)

Figure 5.6: *Horizontal layout irregularities when (a) writing horizontally in the subscript region, (b) writing arguments of fraction bars below the superscript region. The figures at the left show fraction bars in the expression located through MST construction. Dominant baselines are shown at the right.*

Given the symbols s and s_r, an attempt to give a more flexible definition of the right relation is simply to consider s_r as a right horizontal neighbor of s, if s_r lies in the *right* region of s or s lies in the *left* region s_r. This condition is more robust but still restrictive because it only considers symbols having "acceptable" *right* and *left* regions. That is not the case when writing horizontal lines, as we can see in Fig. 5.6. In this case, the right region for the horizontal lines is practically eliminated. Note that the expressions in the figure are written in a regular way.

There are two common horizontal layout irregularities when writing mathematical expressions. One of them occurs when writing horizontal bars below the subscript threshold. For example, the fraction lines in Fig. 5.6(a) satisfy the relation subscript and not the desired relation right horizontal, as defined above. Other irregularities occur when superscripted fractions extend beyond their corresponding superscript region. This is illustrated in Fig. 5.6(b). The fraction argument 'σ' satisfies the relation right with respect to 'e', and not the relation superscript.

To avoid these layout problems, we use an MST construction to re-label horizontal lines into fraction lines and to reconstruct their left and right regions.

MST Construction for Horizontal Bars

To construct the MST, we initialize it with the leftmost symbol in L. Let (s_t, s_n) be a new edge with $s_t \in$ MST and $s_n \notin$ MST. If s_t is a horizontal bar and `dominates`(s_t, s_n) is true, the weight is the minimum distance between the AP. If s_t is a horizontal bar and `dominates`(s_t, s_n) is false, the weight is the minimum distance between the AP of s_t and the centroid of s_n. If s_n is a horizontal, we repeat the calculation as above, considering whether s_n dominates s_t or not. If both s_t and s_n are not horizontal bars, the weight is the distance between their centroids.

Updating Attributes for Horizontal Lines

Once the MST is constructed, we update symbol attributes as follows. Let h be a horizontal bar. Consider also the list Dh, which collects those symbols having edges incident to h and being dominated by h. If symbols in Dh lie in both regions *above* and *below* h, it is re-labelled as fraction bar. If symbols are found only in the *above* region, the horizontal bar is re-labelled as underline. If it is the case for the *below* region, h is re-labelled as over bar. For all the previous cases, lower left corner and height are updated as $y_h = y_{Dh}$ and $H_h = H_{Dh}$, as well as the corresponding derived attributes. Figure 5.6 shows the MST of the expressions and the bounding boxes of the renamed horizontal bars.

Once the attributes of fraction bars are updated, we proceed as follows. Consider again the symbols s and s_r. If s_r is a fraction bar not belonging to the superscript region of s, then s_r is a horizontal right symbol of s if it satisfies the conditions described above. If s_r is a symbol in the right region of s and there is a fraction bar in the superscript region which dominates s_r, then s_r is *not* a horizontal right symbol of s. Finally, suppose s and s_r are two horizontal bars, then s_r is a horizontal right symbol of s if the angle between the x-axis and the segment defined by the centroid of s and the centroid of s_r is less than some threshold.

5.3.2 Construction of the Baseline Tree

Symbol Regions

When we defined the symbol regions in Sect. 5.2.1, we considered them as unbounded regions extending through the plane. Some authors consider the range of symbols lying in the dominant baseline not limited by the threshold attributes alone, but also by "neighbor" operators [103]. They bound the superscript, subscript, and right regions

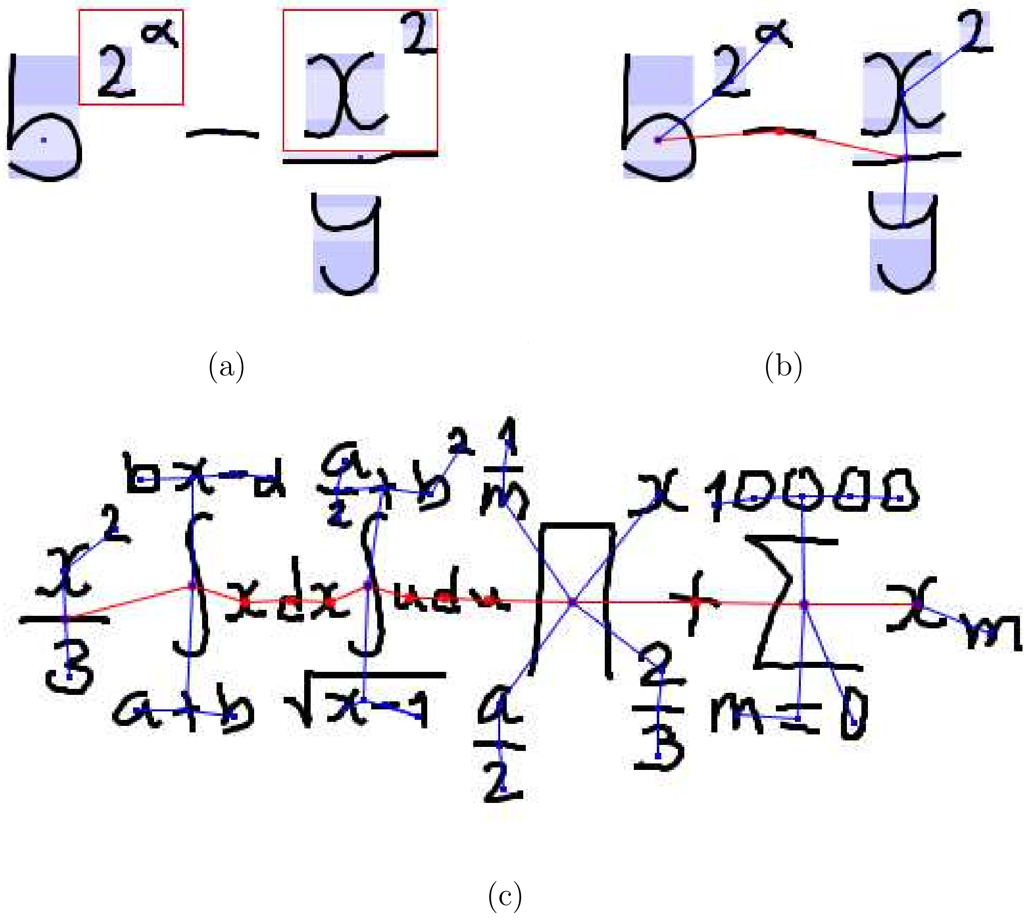


Figure 5.7: *(a) The boxes are the bounded superscript and above regions of 'b' and the fraction bar respectively. (b) The same arguments, 'α' and '2', are written beyond the bounded regions, but they still satisfy the same mathematical relations. (c) A more complicated expression showing the difficulties with sum-like operators.*

at the right by the leftmost x-coordinate of the next right symbol in the baseline. This makes sense because such spatial considerations avoid ambiguity. But these considerations can be very restrictive if we want to correct an expression by adding some superindexes after entering it. The same applies when associating arguments to sum-like operators lying too near to each other and when writing operators like ${}^{*}_{*}\prod^{*}_{*}$ whose arguments have non-standard layouts. Figure. 5.7 shows examples of this.

The key in the MST construction is to associate symbols not only considering where they lie, but also how *near* they lie to each other and how dominance occurs during weight construction.

MST Construction for Symbol Clustering

The function `constructMST`(Db, L) constructs the MST of the symbol list L. We initialize the MST to the dominant baseline Db of L. Let (s_t, s_n) be a new edge with $s_t \in$ MST and $s_n \notin$ MST. The edge weights are computed as follows. If `dominates`(s_t, s_n) is true, the weight $w(s_t, s_n)$ is the minimum distance between *attractor points* of symbols s_t and s_n. If `dominates`(s_t, s_n) is false, the weight corresponds to the distance between the centroids of s_t and s_n. Finally, if the relation `right`(s_t, s_n) or `right`(s_n, s_t) is true, the weight is the minimum distance among their corresponding black points as shown in the second a of Fig. 5.5. Figure 5.7(b)-(c) shows the MST derived by this weight computation.

Arguments to Special Operators

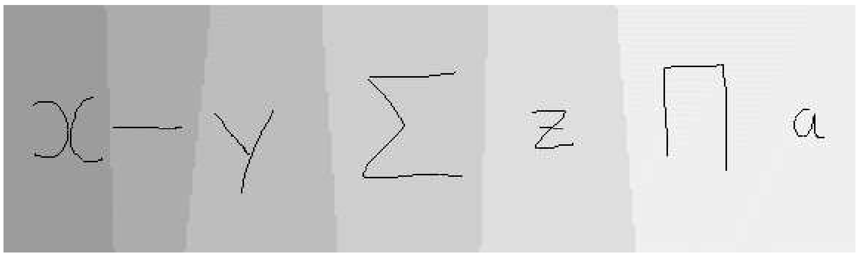
In some way, the MST construction we use for grouping does not bound regions; on the contrary, it makes them grow. To illustrate this, consider Fig. 5.8. It shows how attractor points and symbol dominance help to define dominance regions.

To draw the grey regions in the figure, we proceeded as follows. First, we took a pixel from the whole region and translated the symbol 'a', the leftmost symbol in the figure, so that its centroid and the pixel coincide. Secondly, we associated the symbols of the baseline $(x, -, y, \sum, z, \prod)$ with a grey tone. Finally, the pixel was colored with the grey tone corresponding to the symbol s of the baseline, whose weight $w(s, a)$ reaches the minimum value among all symbols in the baseline. Figure 5.8(a) shows the regions for each symbol, using the distance between centroids as edge weights.

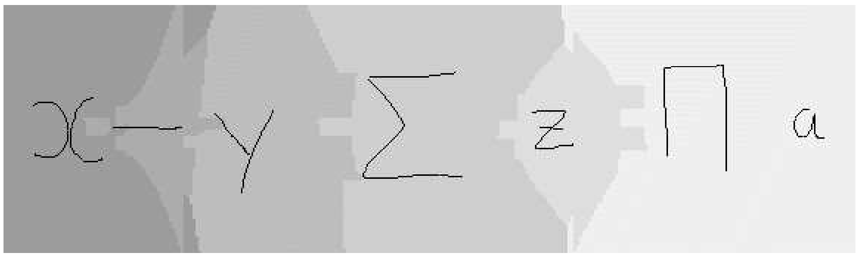
We can appreciate in Fig. 5.8(b) that using symbol dominance to delimit regions corresponds to the expected range of symbol operators. Figure 5.8(d) shows how regions grow during MST construction. In this example, the regions of the symbols z and a are merged in such a way that the new symbol b lies within their range and ambiguities arising from argument association with $\prod$ are overcome. The regions were found as described before, but in this case we used the symbol b instead of a.

5.3.3 Recognition of Matrices

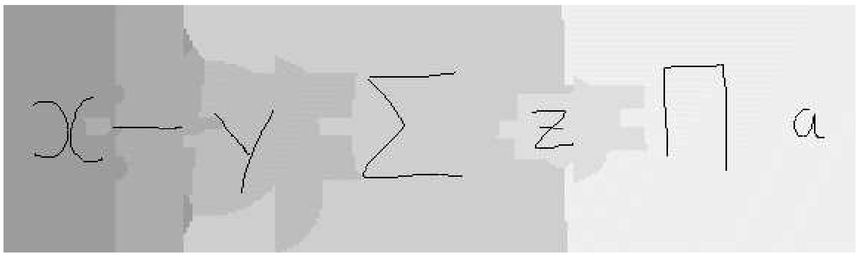
To recognize matrices, we have to define row structures. Note that, in some sense, we defined row structures when locating symbols in subscript, superscript, and right regions. The advantage we have in this case is that the mentioned regions are well defined by the threshold attributes. In contrast, matrices do not have predefined row



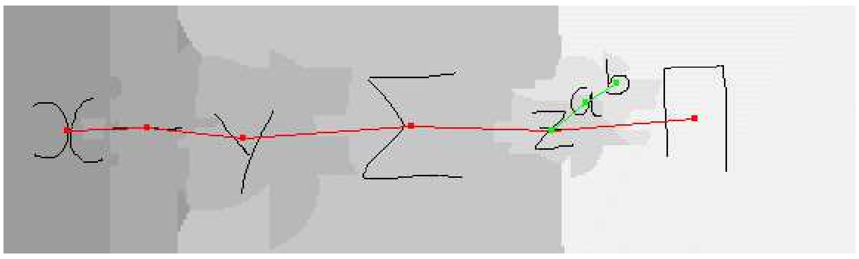
(a)



(b)



(c)



(d)

Figure 5.8: *Regions defined (a) using only the centroids, (b) using the attractor points and symbol dominance without distance factor, and (d) with distance factor. (d) Growing regions in the MST construction.*

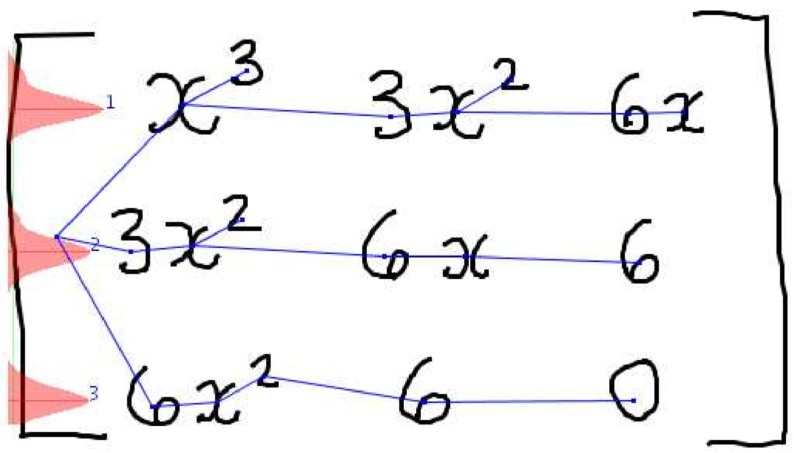


Figure 5.9: *The area-projection function and the MST found in the matrix mode.*

regions, they have to be found on the fly. It is done by dynamically finding APs, as explained in the following section.

MST Construction for Matrices

The symbols '[' and ']' were taken as reserved symbols to construct matrices. The range of the symbol '[' is the bounding box which contains it and its corresponding closing square bracket. Symbols lying in that region are dominated by '[' and are automatically associated to it during MST clustering. Then we check for each $s \in Db$ whether it is an open square bracket or not. If it is, we proceed to identify row structures in the child list Ds of symbols dominated by s. For this purpose, we define the *area-projection function* f as

$$f(y) = \sum_{\substack{s \in Ds \\ y_s \le y \le y_s + H_s}} W_s H_s, \tag{5.1}$$

where $y_{Ds} \le y \le y_{Ds} + H_{Ds}$. We use local maxima of a smoothed version of f, located at y_i, $i = 1, \ldots, n$, to define the attractor points (x_{Ds}, y_i) of s (see Fig. 5.9). The next step is to construct the MST of s and Ds using the dynamically constructed attractor points. Because we want to find rows in the symbol list, we multiply the x-coordinates of attractor points and centroids by a factor $0 < \beta < 1$ and we re-calculate the weights of the graph with this modification. Finally, we assign rows to the corresponding child lists of s, locate spaces in the rows, and apply the method recursively to those lists.

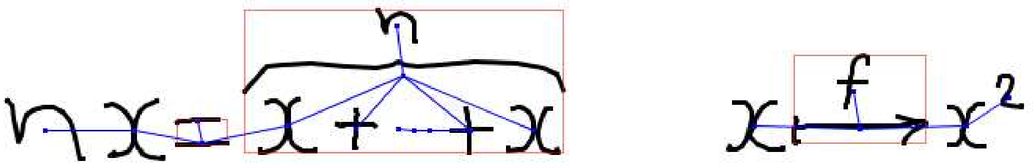


Figure 5.10: *Examples of expressions which can be recognized by the proposed extension for fraction location.*

5.4 Discussion

5.4.1 Extensions

Fraction Location

The preprocessing step for fraction location allows to locate other fraction-like constructions. It can be used to locate and to determine attributes of the symbols like 'overbrace' and 'underbrace'. Other extensions could be the location of function names by using "arrow notation" for this purpose. See Fig. 5.10.

Symbol Clustering

The MST construction for symbol clustering can be easily extended to recognize expressions that contain, for example, operators like ${}_n\mathbf{C}_k$, or other operators with similar layouts. It can be also extended to recognize not only square roots but also the structure $\sqrt[*]{\ }$.

Matrix Recognition

The method for the recognition of matrices can be easily extended to recognize stacked arguments of sum-like operators, like the one used in (5.1). The tabular mode can be set by default when analyzing the subscript list of sum-like operators. This MST construction can also be useful to recognize structures which describe equation systems as well as functions defined by cases. For these structures, we can use the symbol '{' as indicator of the tabular mode. See Fig.5.11

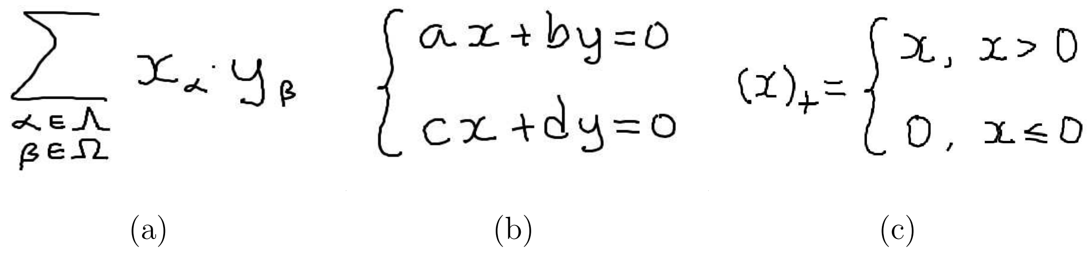


Figure 5.11: *Examples of expressions which can be recognized by the proposed extension for matrix recognition.*

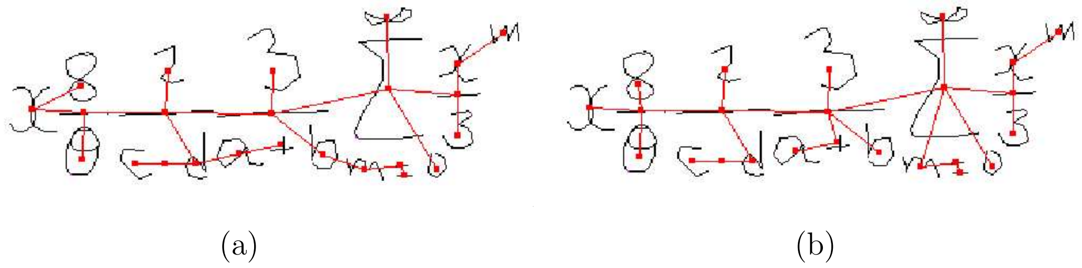


Figure 5.12: *The result of the MST construction (a) without using the α factor in weight calculation and (b) using this factor.*

5.4.2 Limitations of the Method

Symbol Grouping

Our method encounters problems when scripted symbols lie too close to the arguments of fraction or sum-like operators. The horizontal baselines of dominated symbols are merged incorrectly when they are written too far away from operators and the latter are written too close to each other. See Fig. 5.8(c) and Fig. 5.12(a) for an example of this. To avoid the problem, we multiply the corresponding weight by a factor $0 < \alpha < 1$ during MST construction if the symbol in the dominant baseline is a sum-like operator or a fraction line. See Fig. 5.12(b).

Parameter Estimation

Determining heuristic values for α and β requires some experimentation. We have obtained satisfactory results by using the values $\alpha = 1/4$ and $\beta = 1/15$. We plan to construct a benchmark of on-line handwritten mathematical expression to determine the optimal values of the parameters α and β and others required by our algorithm, as well as to obtain a precise estimation of recognition rates.

Symbol Recognition

The results of the previous chapter showed that classification errors can occur. This is a serious problem when incorrectly recognizing key operators, for example '$\int$' as '5', because they are crucial for baseline construction. Recall that our method assumes perfect recognition rates. To avoid problems generated by misrecognized symbols, we can use an interface which allows immediate feedback and has undo-redo and visualization capabilities, as we will see in the following chapter.