# *ExpressMath*: Análise estrutural de expressões matemáticas manuscritas

Ricardo Sider
Bruno Yoiti Ozahata

Nina S. T. Hirata (orientadora)

Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo

1 de Dezembro de 2009

**Resumo**

Esta monografia descreve o trabalho desenvolvido como parte dos requisitos para a disciplina MAC0499 - Trabalho de Formatura Supervisionado do curso de Bacharelado em Ciência da Computação. Apesar do reconhecimento de escrita já ser um tema de pesquisa bastante explorado, ainda existem muitos desafios, um dos quais é o reconhecimento de notação matemática, um caso em que a quantidade de possíveis símbolos é grande e a disposição espacial dos símbolos é bastante complexa. O reconhecimento de expressões matemáticas é geralmente dividido em duas etapas: (1) segmentação e reconhecimento dos símbolos, e (2) análise estrutural. Neste trabalho apresentamos uma abordagem encontrada na literatura recente para análise estrutural de expressões matemáticas manuscritas. A abordagem explora conceitos como *baselines* e árvores geradoras mínimas para determinar a estrutura de uma expressão. Descrevemos também uma implementação dessa abordagem. Durante parte do desenvolvimento deste trabalho, os autores receberam bolsa do CNPq.

# Conteúdo

# Parte I

# Parte Técnica

## 1 Introdução

Expressões matemáticas transmitem informação por meio de um arranjo bidimensional de símbolos, e seu reconhecimento automatizado é um ramo do reconhecimento de padrões de extrema importância prática. Seus usos vão da conversão de artigos e livros científicos para o formato digital a fácil criação de documentos em LaTeX sem conhecimento dos comandos específicos da linguagem para a inserção de símbolos matemáticos. Além disso, o reconhecimento pode ser útil para auxiliar portadores de deficiência visual, através de "leitura" de anotações escritas em quadros ou papéis. Pode também simplificar a entrada de dados em softwares especializados em manipulação algébrica de expressões matemáticas, entre outras utilidades [1, 2, 7].

A linguagem matemática é especialmente complicada no que diz respeito ao seu reconhecimento, pois aceita inúmeros dialetos, com uma grande variedade de símbolos. Além disso, as relações espaciais e de tamanho entre os símbolos são fortemente exploradas nas fórmulas, dando sentidos diferentes às expressões. No caso de expressões manuscritas, em especial, o mau posicionamento dos símbolos é comum, dificultando o trabalho de sistemas de reconhecimento.

Expressões matemáticas podem ser encontradas tipografadas (escritas em um computador) ou ainda manuscritas. Além disso, o reconhecimento pode ser *online* ou *offline*. No segundo, os dados são obtidos após a escrita, por exemplo através de um *scanner*. Já no caso do reconhecimento *online*, a expressão matemática é lida ao mesmo tempo em que o usuário a escreve, possibilitando aos reconhecedores o uso de informações temporais referentes a cada traço.

O reconhecimento e digitalização de expressões matemáticas se divide naturalmente em duas etapas [1, 2]. A primeira consiste na classificação dos símbolos da expressão [12, 14], na qual cada símbolo é isolado em um retângulo envoltório (*bounding box*), a saber, a menor caixa retangular na qual o objeto pode ser colocado, e reconhecido por meio de um classificador, recebendo uma etiqueta que define sua identidade.

Na segunda etapa, é feita a análise estrutural da expressão [6, 13, 15]. Toma-se a lista de símbolos classificados e a posição de seus retângulos envoltórios e analisa-se a relação espacial entre eles. A expressão reconhecida é devolvida em alguma forma estruturada e hierárquica, ou até mesmo reescrita numa linguagem que aceite fórmulas matemáticas, como o LaTeX.

O objetivo desse trabalho é investigar e implementar um algoritmo para análise estrutural de expressões matemáticas manuscritas. Este trabalho está inserido no contexto do projeto *ExpressMath* [8] e dá sequência a trabalhos previamente desenvolvidos [4, 5]. Dentre os trabalhos relacionados à análise estrutural de expressões matemáticas manuscritas existentes na literatura da área, três merecem destaque e foram os que mais influenciaram o desenvolvimento deste

trabalho.

Matsakis [9] foi o primeiro a utilizar uma árvore geradora mínima como passo inicial da análise estrutural. Seu método considera cada traço como um vértice em um grafo totalmente conexo, agrupando-os depois em símbolos, a partir dos quais constrói-se uma árvore geradora mínima que os conecta. Essa árvore representa uma aproximação da estrutura final da expressão. Infelizmente esse método não funciona muito bem para expressões matemáticas complexas.

Zanibbi [15] propôs a construção de uma *Baseline Structure Tree*, uma estrutura baseada no conceito de *baselines* de uma expressão. Seu método corrige irregularidades horizontais na escrita e a sobreposição de símbolos, explorando bastante o conceito de dominância e a ordem de leitura da esquerda para a direita.

Tapia [11] combinou essas duas abordagens, apresentando um sistema que trata de irregularidades horizontais, associação de operadores, e sobreposição de símbolos. Apresenta limitações no reconhecimento de algumas estruturas.

Desta forma, investigamos e implementamos a abordagem proposta em [11]. Este texto está organizado da seguinte forma. Na seção 2 apresentamos os conceitos básicos, tais como dominância entre símbolos e *baselines* de uma expressão, necessários para o entendimento do algoritmo de análise estrutural. Na seção 3 descrevemos o algoritmo para análise estrutural da expressão, que visa construir uma árvore representativa da expressão. Essa estrutura abriga todos os símbolos da expressão e suas relações espaciais, e a partir dela é possível reconstruir facilmente a expressão original em qualquer linguagem digital estruturada. Em seguida, na seção 4, descrevemos a implementação do algoritmo e na seção 5, apresentamos alguns resultados obtidos pelo algoritmo implementado. Apresentamos as conclusões na seção 6.

# 2 Conceitos básicos

Nesta seção descrevemos alguns dos principais conceitos e estruturas de dados utilizadas na abordagem utilizada para a análise estrutural.

## 2.1 Regiões e atributos de símbolos matemáticos

Uma expressão matemática consiste de um arranjo espacial de diversos símbolos no plano bidimensional. A posição relativa entre os símbolos é determinante na definição da relação entre eles e, consequentemente, na definição de suas funções do ponto de vista semântico na expressão.

As posições relativas entre símbolos podem ser caracterizadas em termos de regiões definidas ao redor dos mesmos. Analisando um símbolo, podemos ter outros nas regiões "acima", "abaixo", "superscript", "subscript", "direita", "abaixo e à esquerda", "acima e à esquerda", e "sub-expressão", regiões estas que são usadas para expressar as relações entre símbolos numa expressão.

A quantidade e localização dessas regiões é dependente do tipo de símbolo em questão. Essa

dependência pode ser observada, por exemplo, na diferença da posição do índice nos símbolos $x_*$ e $y_*$. Alguns símbolos, como um operador de divisão por exemplo, não somente possuem regiões onde podem existir atributos, mas exigem a presença dos mesmos (não faz sentido um operador binário sem seus operandos). A figura 1 mostra alguns símbolos e regiões esperadas dos seus argumentos.

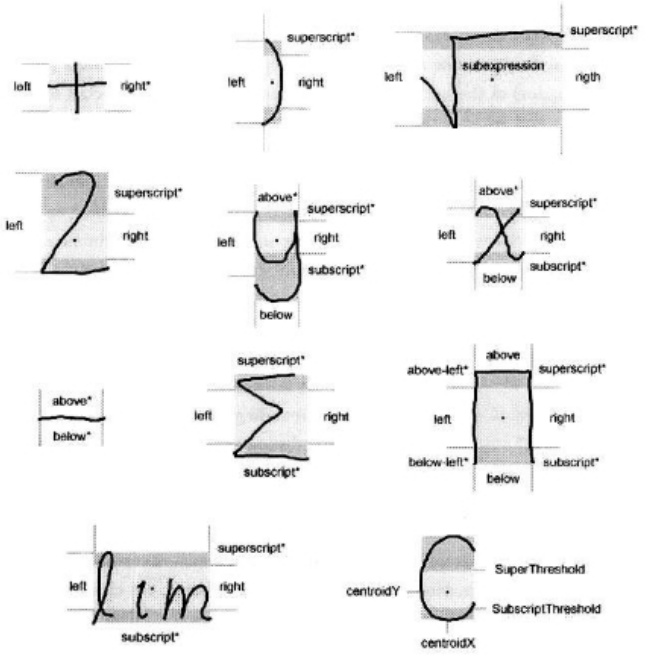



Figura 1: Alguns símbolos com seus atributos e regiões (Fonte: [11])

Para auxiliar na definição das posições dessas regiões espaciais em torno dos símbolos, usamos seus atributos básicos (a etiqueta e o retângulo envoltório) para inferir o centróide e os limiares inferior e superior de um símbolo.

- **Centróide:** representa o seu centro de massa, ou seja, o ponto que poderia substituir o símbolo inteiro durante a análise com perda mínima de significado.
- **Limiares inferior e superior:** são atributos numéricos, que representam limitações para regiões ao redor de um símbolo no eixo vertical. Por exemplo: definimos que um símbolo `b` está à direita do símbolo `a` se a coordenada y do seu centróide está entre os limiares inferior e superior do símbolo `a`.

Esses atributos são calculados em função das coordenadas do retângulo envoltório, e dependem do tipo de símbolo em questão, como será explicado posteriormente. Podemos inferir as relações espaciais entre dois símbolos comparando a posição dos seus centróides e limiares, que por sua vez definem a estrutura de uma expressão matemática.

Na figura 2 vemos o centróide e os limiares de três símbolos com distribuições espaciais diferentes dos traços. Como podemos ver, o cálculo desses atributos sugere a separação dos símbolos em classes, como visto na próxima seção.

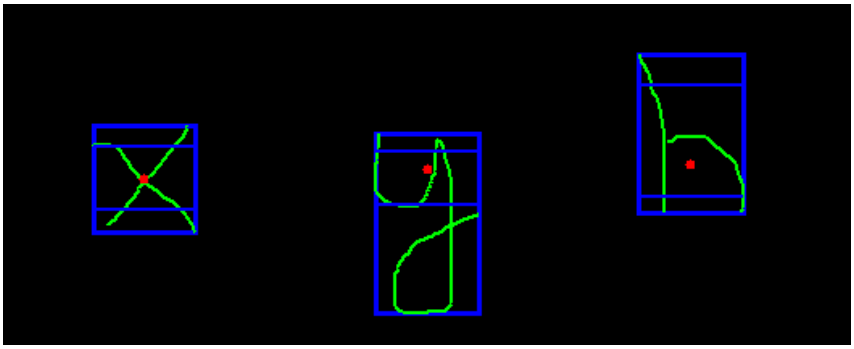


Figura 2: Alguns símbolos com retângulo envoltório, centróide (em vemelho), e limiares superior e inferior

## 2.2 Categoria dos símbolos

Numa expressão matemática podemos encontrar diferentes tipos de símbolos, sejam operadores, letras especiais como o símbolo dos números reais ($\mathbb{R}$), ou até letras do alfabeto grego. Essa ampla gama de símbolos, no entanto, pode ser classificada de acordo com o tamanho, número de operadores, tipo e posição esperada dos argumentos, posição do centróide, entre outras. A separação dos símbolos reconhecidos de acordo com esses atributos é importante no sentido de aumentar a robustez do algoritmo, possibilitando inclusive o reconhecimento correto de expressões ambíguas.

O analisador estrutural pode eliminar alguns arranjos dos símbolos, que embora sejam possíveis de se inferir da entrada do usuário, não fazem sentido do ponto de vista matemático. A figura 3 mostra um exemplo disso.

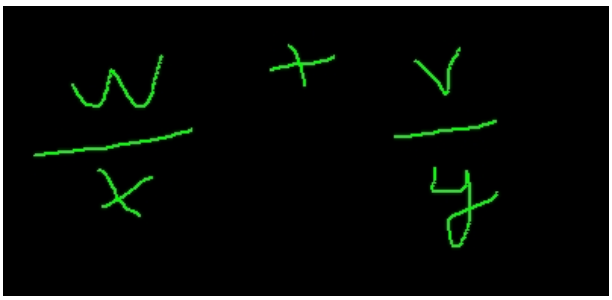


Figura 3: Operador da adição escrito na região de superscript da fração.

### 2.2.1 Segundo sua estrutura geométrica

Internamente, cada símbolo é classificado de acordo com a concentração de traços dentro do seu retângulo envoltório, e a posição esperada de seus argumentos numa expressão, a saber, o centróide e os limiares superior e inferior.

Nesse quesito, cada símbolo pode ser classificado como Central, Ascendente ou Descendente. Essa classificação é interna a cada símbolo, e determina a fórmula para o cálculo do centróide e dos limiares superior e inferior, e também como são realizadas as comparações do tamanho relativo.

A divisão dos símbolos nessas três classes faz bastante diferença na decisão das posições relativas entre eles, eliminando arranjos inviáveis matematicamente, e possibilitando inclusive uma certa tolerância a imperfeições da escrita do usuário, uma qualidade muito importante na análise de expressões manuscritas.

### 2.2.2 Segundo seu aspecto comportamental

A divisão dos símbolos em categorias tem o objetivo de limitar o comportamento de acordo com características matemáticas específicas. Operadores binários, por exemplo, se comportam de uma maneira bastante diferente que um operador com limites, ou uma raiz.

Podemos definir quantas classes julgarmos necessárias para definir o comportamento dos símbolos. Num extremo podemos considerar cada símbolo uma classe por si só, com comportamento altamente específico. No entanto, essa abordagem implica uma quantidade muito alta de classes, dificultando o entendimento e a análise do código produzido. No ExpressMath optamos por dividir os símbolos em seis classes, agrupando símbolos com comportamento semelhante:

- **Raiz:** O operador da raíz foi separado dos outros, pois é o único símbolo que pode ter outros símbolos contidos nele, e sua região superior esquerda tem um significado especial, tanto matematicamente (radicando), quanto na sua representação em LaTeX. A posição dos seus operadores é limitada a uma sub-expressão (obrigatória), um possível radicando, subscritos e superscritos.

- **Barra horizontal:** A barra horizontal é um símbolo que merece uma classe exclusiva, pois pode assumir o papel de diferentes símbolos dependendo dos argumentos encontrados: operador de divisão, sublinhado, ou sobrelinha. Além disso, cuidado especial é necessário ao definir a dominância, visto que esse operador a exerce verticalmente, e embora seja de tamanho insignificante no eixo vertical, domina sobre seus operandos, não importando seu tamanho. Também como decorrência do seu aspecto achatado, precisamos tomar cuidados especiais para definir a sua região adjacente no eixo horizontal.

- **Operadores com Limites:** Podem ter atributos superiores e inferiores apenas. Cuidado especial é necessário quanto seus argumentos invadem as regiões à esquerda e à direita.

Também é necessário tratar o caso em que seus argumentos são escritos longe do operador, muito comum em expressões matemáticas manuscritas. Esse tratamento será melhor explicado posteriormente.

- **Scripted:** Letras, números e símbolos que tem significado completo sozinhos. A princípio podem ter argumentos em qualquer região espacial, exceto acima e abaixo.
- **Non-scripted:** Operadores unários ou binários e símbolos matemáticos diversos. A princípio só possuem argumentos os operadores matemáticos.

Embora essa classificação seja comportamental, podemos usá-la também para o cálculo dos limiares. Programaticamente cada classe de símbolos é uma implementação diferente de uma mesma interface, de modo que podemos sobreescrever qualquer método que define comportamento.

É importante ressaltar que em alguns casos há também limitação para o tipo dos argumentos esperados. Por exemplo, embora um símbolo da classe *Scripted* permita um argumento subescrito, não faz sentido que esse argumento seja um operador binário.

### 2.2.3 Desvantagens

Embora a classificação prévia dos símbolos nas classes descritas acima seja muito vantajosa nas decisões tomadas no algoritmo, ela gera também o aspecto negativo de limitação dos símbolos aceitos. Dada uma etiqueta, o ExpressMath deve saber reconhecê-la para classificar o símbolo. Para minimizar esse problema, buscamos manter uma lista ampla de símbolos aceitos, contendo a maioria dos símbolos relevantes aceitos no LaTeX.

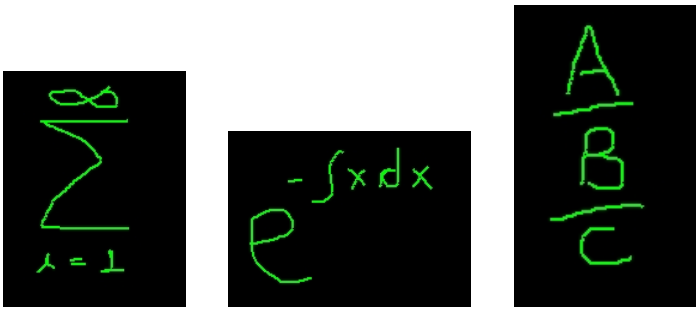
## 2.3 Dominância

Símbolos em uma expressão matemática são agrupados segundo uma certa hierarquia, definida implícita ou explicitamente pela posição e tamanho relativo dos símbolos na expressão [3]. Uma barra de divisão, por exemplo, é mais relevante que seus operandos (numerador e denominador). Dizemos portanto que um operador exerce dominância sobre seus argumentos. Assim, podemos definir uma função booleana, `domina(a, b)`, que é verdadeira quando o símbolo **a** exerce relação de dominância para com o símbolo **b**.

A implementação dessa função leva em consideração o tipo do símbolo, e as áreas relativas a ele nas quais parâmetros são esperados. Voltando ao exemplo do operador da divisão, temos as regiões superior e inferior, nas quais não só esperamos outros símbolos, como os demandamos.

Definimos a abrangência de um símbolo como toda a área onde esperamos encontrar operandos e outros atributos dominados por ele. A partir dessa definição, podemos dizer que um símbolo **a** domina sobre outro símbolo **b**, se **b** está na área de abrangência de **a**, e **a** não está na área de abrangência de **b**.

Em casos em que a dominância não pode ser determinada dessa forma, existem ainda outros critérios, tais como tamanho relativo dos símbolos, e a ordem de leitura da esquerda para a direita (um símbolo mais a esquerda domina o próximo símbolo).




(a) (b) (c)

Figura 4: Expressões onde (a) Dominância é determinada pela abrangência. (b) Dominância é determinada pelo tamanho. (c) É difícil determinar a dominância dos operadores

No caso da figura 4(c), podemos resolver a ambiguidade definindo a dominância do operador de divisão de maior comprimento, ou até do de menor coordenada y. Esse exemplo mostra como o conceito de dominância está bastante aberto a interpretações. Diferentes definições desse conceito ajudam a definir dialetos da linguagem matemática, e a literatura sobre este tema apresenta diversas definições desse conceito.

## 2.4 Baselines

Apoiados no conceito de dominância entre símbolos, podemos passar a explorar formas de representar uma expressão matemática em função dos símbolos dominantes e de seus atributos. Definimos uma *baseline* como uma lista de símbolos que representa um arranjo horizontal na expressão. Desse modo, podemos representar uma fórmula matemática como um conjunto de baselines aninhadas hierarquicamente [15]. Cada símbolo pode ter ligação para outras baselines, que satisfaçam relações espaciais específicas para com ele. A expressão $x_i + y^2 + \frac{a+b}{d}$, por exemplo, é composta por 5 baselines, [x, +, y, +, -], [i], [2], [a, +, b] e [d], sendo que as 4 últimas estão ligadas aos símbolos x (subscript), y (superscript), - (acima), e - (abaixo).

À *baseline* que não está associada a nenhum símbolo damos o nome de *baseline* principal da expressão, e todas as outras baselines estão ligadas e ela, direta ou indiretamente, por conexões rotuladas com a relação espacial satisfeita entre elas.

A figura 5 mostra uma expressão matemática, com suas baselines em vermelho.

## 2.5 Árvore Geradora Mínima

Uma árvore geradora de um grafo é um subconjunto de suas arestas, de modo que todos os vértices possam ser atingidos por meio dela. Quando trabalhamos com um grafo com pesos nas

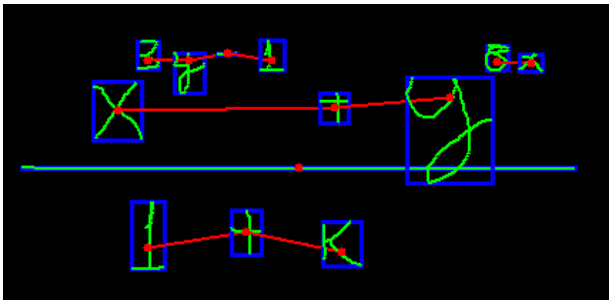


Figura 5: Baselines de uma expressão matemática

arestas, denominamos árvore geradora mínima a árvore geradora cuja soma dos pesos de suas arestas é mínimo.

Em nossa abordagem consideramos os símbolos de uma fórmula matemática como vértices de um grafo totalmente conexo e podemos atribuir pesos às suas arestas de acordo com a proximidade e relações de dominância entre símbolos. Desse modo, pode-se construir a árvore geradora mínima desse grafo, estrutura que nos dá uma boa aproximação das relações espaciais entre os símbolos na expressão.

## 3 Algoritmo para análise estrutural

No ExpressMath, o ponto de partida para a análise estrutural de uma expressão matemática é uma lista de símbolos previamente reconhecida pelo classificador. Cada item dessa lista contém informações básicas sobre o símbolo, a saber, sua classificação, o conjunto de seus traços (com informação de tempo), e seu retângulo envoltório (*bounding box*).

Para que seja possível dar início ao algoritmo de análise estrutural, cada símbolo dessa lista é classificado em dois tipos de categorias: uma relacionada à disposição geométrica dos traços dentro do retângulo envoltório, e outra relacionada ao comportamento esperado daquele símbolo numa expressão matemática, conforme explicado anteriormente.

A partir dessa lista de símbolos classificados dá-se início ao processo de análise estrutural, seguindo em linhas gerais o seguinte fluxo:

1. Determinar *baseline* principal;
2. Calcular a árvore geradora mínima;
3. Construir listas com os filhos dos símbolos na *baseline* principal, rotuladas com a relação espacial que satisfazem com ele;
4. Aplicar a recursão para cada uma das listas não vazias.

## 3.1 Determinar a *baseline* principal

A construção da *baseline* principal de uma expressão matemática explora bastante o conceito de dominância e a ordem de leitura da esquerda para a direita. A seguir explicamos duas sub-rotinas básicas para sua construção, e em seguida a função que os combina recursivamente, construindo a *baseline*.

### 3.1.1 Encontrar o primeiro símbolo dominante mais à esquerda

Para encontrar o primeiro símbolo dominante de uma expressão utilizamos uma função simples, que vai removendo símbolos não dominantes do final da expressão, até que só reste o símbolo desejado. A seguir mostramos o pseudo-código dessa função.

```
Símbolo start(Lista lista)
    se |lista| = 1
        devolva o único elemento de lista

    ultimo = remova último de lista
    penultimo = remova último de lista

    se domina(ultimo, penultimo)
        adicione ultimo em lista
    senão
        adicione penultimo em lista
```

### 3.1.2 Encontrar vizinhos à direita

Conhecendo o primeiro símbolo dominante da expressão podemos usar uma função que encontra todos os seus vizinhos à direita, retornando uma lista dos símbolos considerados adjacentes ao primeiro dominante. A chave para um reconhecimento satisfatório da *baseline* principal é apresentar uma definição robusta de adjacência. Cuidado especial é necessário no tratamento de irregularidades horizontais, bastante comuns em expressões manuscritas.

### 3.1.3 Algoritmo

```
Baseline construirBaselineDominante (Lista lista)
    se |baseline| = 0
        primeiro = start(lista)
        inserir primeiro em baseline

    vizinhos = encontrarVizinhosADireita(lista, baseline.pegarUltimo())

    se |vizinhos| = 0
        devolva baseline

    senão
        next = start(vizinhos)
        inserir next em baseline

        devolva construirBaselineDominante(lista);
```

### 3.1.4 Comprimir símbolos compostos

Após construir a *baseline* principal, verificamos se existem nela símbolos que agrupados representem algum outro símbolo aceito pelo LaTeX como por exemplo, `cos`, `sen`, `tan`, `lim` e entre outros.

Caso algum desses padrões seja encontrado, agrupamos os símbolos correspondentes, removendo-os da estrutura e adicionando o novo símbolo agrupado com uma nova etiqueta e atributos recalculados (caixa envoltória, centróides e limiares).

## 3.2 Construção da árvore geradora mínima

A construção da árvore geradora mínima da expressão é o passo mais importante na abordagem proposta. É essa etapa que irá determinar quais são as relações espaciais mais relevantes na expressão, definindo a sua estrutura. Nas próximas seções apresentamos a proposta da abordagem, e em seguida discutimos alguns dos detalhes e cuidados necessários.

### 3.2.1 Proposta

Após o reconhecimento da *baseline* principal, construímos a árvore geradora mínima, também chamada de *Minimal Spanning Tree* (MST) da expressão. Para isso, utilizamos o algoritmo de Prim [10]. Tal algoritmo utiliza do conceito de franja, que pode ser definido como um conjunto de arestas de um grafo `G` com um vértice pertencente a árvore `T` e a outra ponta fora desta. Na primeira iteração do algoritmo, a árvore `T` é um vértice qualquer do grafo, a partir da qual

procura-se na sua franja a aresta de custo mínimo que será adicionada à árvore `T`. Enquanto a franja de `T` não for vazia, repete-se esse processo. Se o grafo `G` for conexo, o algoritmo encontra uma arvore geradora mínima, caso contrário, encontra uma árvore de uma componente de `G`.

Na implementação apresentada neste trabalho, os símbolos da expressão matemática são representadas por vértices de um grafo totalmente conexo.

### 3.2.2 Cálculo dos Pesos

Na construção da MST, o passo mais importante é o cálculo dos pesos das arestas, pois terá total influência na estrutura final da árvore, e consequentemente, da expressão. Esse cálculo é feito com base na distância real entre os símbolos, participação em uma *baseline* pré-reconhecida, e principalmente dominância entre símbolos.

Dados dois símbolos $a$ e $b$, o cálculo do peso da aresta que os une ocorre da seguinte maneira:

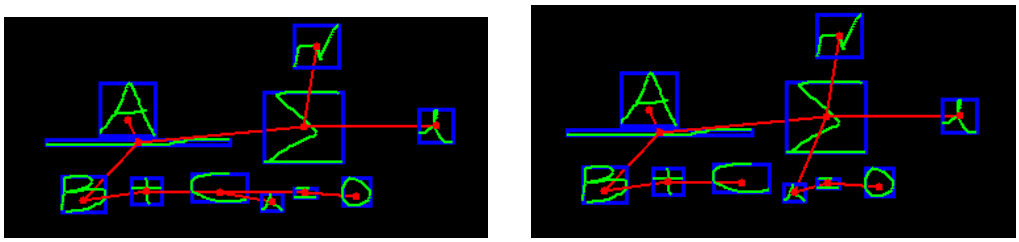
- **Caso 1** Se não existem relações de dominância entre os símbolos, o custo da aresta será a distância entre seus centróides.
- **Caso 2** Se `domina(a, b)`, ou `domina(b, a)`, o custo será a menor distância entre seus pontos de atração.

Os pontos de atração, ou *attractor points*, são pontos localizados nas bordas do retângulo envoltório de cada símbolo, e seu número e posição variam de acordo com sua classe. Além disso, se um símbolo faz parte da baseline principal da expressão sendo analisada, posicionamos mais pontos de atração, de modo a minimizar o custo final das arestas que ligam símbolos na *baseline* a outros símbolos fora dela.

Em seu trabalho, Tapia [11] não informa o número ou posição ideal dos pontos de atração, e é aceitável ponderar que não se possa chegar a um arranjo que seja absolutamente melhor que outro. Assim, a escolha desses pontos ocorreu empiricamente no ExpressMath, com o objetivo de minimizar as taxas de erro para o conjunto de testes utilizado.

### 3.2.3 Cuidados especiais

Usando essa técnica para o cálculo dos pesos conseguimos obter resultados satisfatórios para a maioria das expressões. No entanto, uma construção recorrente em expressões matemáticas manuscritas que pode não ser reconhecida corretamente é a escrita dos limites de símbolos muito longe do seus operadores. Havendo algum outro símbolo próximo, os limites serão associados com ele, gerando uma expressão inválida. Para corrigir esse problema, multiplicamos o peso da arestas que ligam um operador com limites ao seu argumentos com um fator $0 < \alpha < 1$. A figura 3.2.3 mostra um exemplo de reconhecimento com e sem esse fator.



(a) (b)

Figura 6: Árvore geradora mínima da expressão (a) Sem o fator $\alpha$. (b) Usando o fator $\alpha$.

### 3.3 Obtenção das novas listas de filhos

Após a obtenção da árvore geradora mínima da expressão, percorre-se a *baseline* principal, verificando para cada símbolo se existem arestas ligando-a a outros vértices. Para cada aresta saindo da *baseline* principal criamos uma nova lista contendo todos os símbolos que se conectam à *baseline* principal por meio dela. Essa lista é identificada pela relação espacial que satisfaz para com o seu símbolo pai na *baseline* principal.

Após ordenar essas listas em ordem crescente da menor coordenada horizontal dos símbolos, aplicamos o algoritmo de análise estrutural recursivamente.

## 4 Implementação

O algoritmo de análise estrutural descrito na seção anterior foi implementado e integrado ao sistema ExpressMath. O sistema ExpressMath é uma evolução do Math-Picasso [5], e está sendo desenvolvido em conjunto com uma outra equipe. O ExpressMath aproveitou parte do código do Math-Picasso, principalmente na parte relativa à aquisição dos traços da escrita e à segmentação dos símbolos (agrupamento dos traços em símbolos individuais). A outra equipe foi responsável por algumas reestruturações no código e está responsável pela introdução de melhorias na parte de reconhecimento (classificação) dos símbolos. O ExpressMath deverá ser posteriormente disponibilizado como software livre.

O ExpressMath foi desenvolvido em Java, de acordo com o padrão *Model View Controller* (MVC) de arquitetura de software. Essa estrutura visa minimizar o acoplamento entre a interface com o usuário e o resto do código, facilitando alterações em ambos.

### 4.1 Reconhecimento

O ExpressMath executa integralmente as duas etapas do reconhecimento de expressões matemáticas, sendo a primeira dividida em duas sub-etapas:

- **Captura da escrita:** captura dos traços de entrada e sua segmentação em símbolos.

A interface de captura, e o algoritmo de segmentação foram integralmente herdados do Math-Picasso.

- **Classificação dos símbolos:** reconhecimento dos símbolos da entrada, associando cada um à sua representação em LaTeX. Corresponde ao trabalho sendo desenvolvido pela outra equipe.
- **Análise estrutural:** a partir dos caracteres devidamente classificados, é realizado o reconhecimento estrutural e gerado o código LaTeX da expressão.

Uma vez que a abordagem de análise estrutural investigada nesse trabalho requer que os caracteres e símbolos da expressão estejam previamente classificados, decidiu-se por uma implementação na qual a parte de análise estrutural pudesse ser desenvolvida e testada de forma independente da parte de classificação. É muito importante para o nosso método que os caracteres estejam reconhecidos de maneira correta, pois as diferenças comportamentais entre "5" e "$\int$", por exemplo, causariam comportamento inesperado do algoritmo.

Esse resultado foi atingido por meio de alterações na interface do Math-Picasso [5], feitas em conjunto com o grupo que está trabalhando no classificador. Em particular, disponibilizamos uma interface para classificação manual dos símbolos, na qual o usuário digita o código LaTeX correspondente à expressão. A figura 7 mostra um exemplo de uso desse classificador manual.

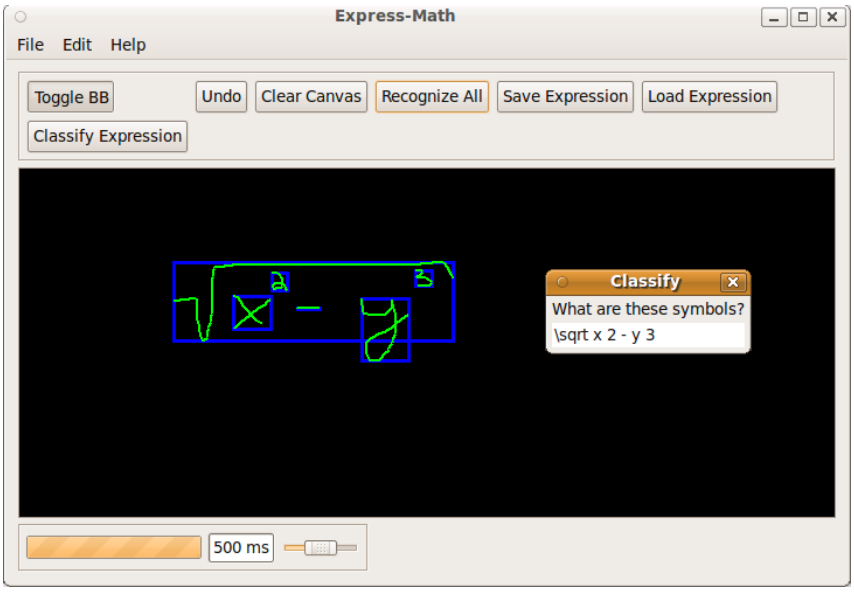



Figura 7: Exemplo de uso do reconhecedor manual.

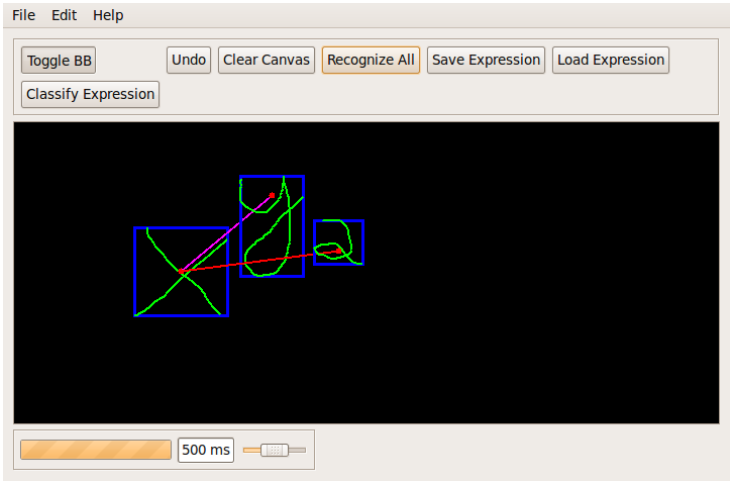
## 4.2 Interface de correção de erros

Como no reconhecimento computadorizado de linguagens naturais não é possível obter um sistema 100% eficiente, uma das preocupações durante o desenvolvimento foi disponibilizar uma

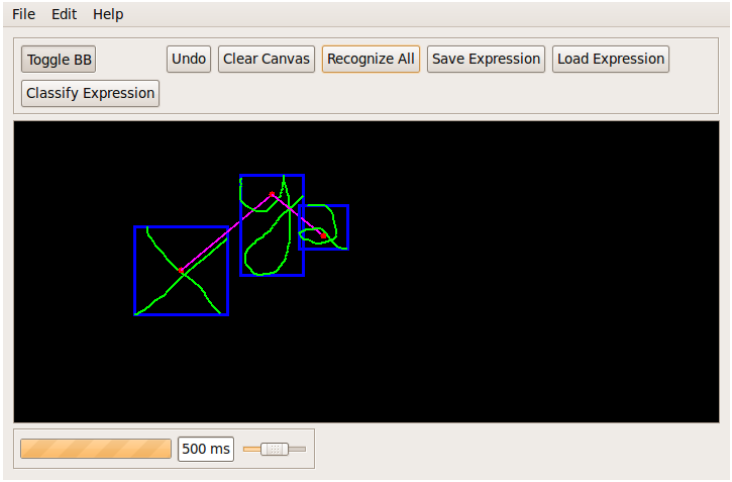
interface amigável, que permitisse ao usuário corrigir, em tempo de execução, possíveis erros cometidos pelo programa.

A principal preocupação no desenvolvimento dessa interface foi a simplicidade e usabilidade. A maior dificuldade em atingir esse objetivo foi evitar que a interface exigisse conhecimento do usuário sobre o funcionamento do algoritmo, ou seja, conhecer conceitos como *baseline* principal, ou árvore geradora mínima.

A solução encontrada foi uma interface no estilo *drag & drop*, na qual o usuário pode arrastar símbolos na expressão, de modo a priorizar as relações desejadas. Essa abordagem se mostrou bastante adequada a usuários sem conhecimento prévio do algoritmo. A figura 8 mostra um exemplo de uso dessa ferramenta.




(a)




(b)

Figura 8: (a) Expressão reconhecida equivocadamente: $x^y 2$. (b) Expressão corrigida com a interface *drag & drop*: $x^{y_2}$.

A introdução desse método de correção induziu uma nova etapa na avaliação de desempenho

do algoritmo: determinar o número de movimentações necessárias para corrigir uma expressão equivocadamente analisada, ou a impossibilidade dessa correção. Os resultados obtidos nessa etapa serão apresentados na seção 5.

A figura 9 apresenta telas do nosso sistema em diferentes estágios do reconhecimento.

### 4.3 Formato de Saída

O resultado final do processo de análise estrutural do nosso sistema é a árvore geradora mínima da expressão. Nela, cada símbolo é um vértice que possui no máximo 8 arestas, representativas das possíveis relações entre os símbolos que ligam ("acima", "abaixo", "superscript", "subscript", "direita", "abaixo e à esquerda", "acima e à esquerda", e "sub-expressão"). Raramente, no entanto, um símbolo gera mais de 3 arestas.

É fácil ver que essa estrutura pode ser percorrida facilmente de maneira recursiva, possibilitando a reconstrução da expressão por meio de um algoritmo razoavelmente simples, em qualquer formato digital estruturado.

Para este trabalho, escolhemos o LaTeX para representar a expressão final. Essa escolha levou em consideração a ampla aceitação dessa linguagem para produção de textos científicos no meio acadêmico, a abertura de seu código e especificações, a ampla gama de símbolos reconhecidos, e também fatores limitantes de dialeto, conforme explicados anteriormente. O LaTeXinfluenciou na definição das classes de símbolos, uma vez que seu comportamento foi limitado ao permitido por esse formato.

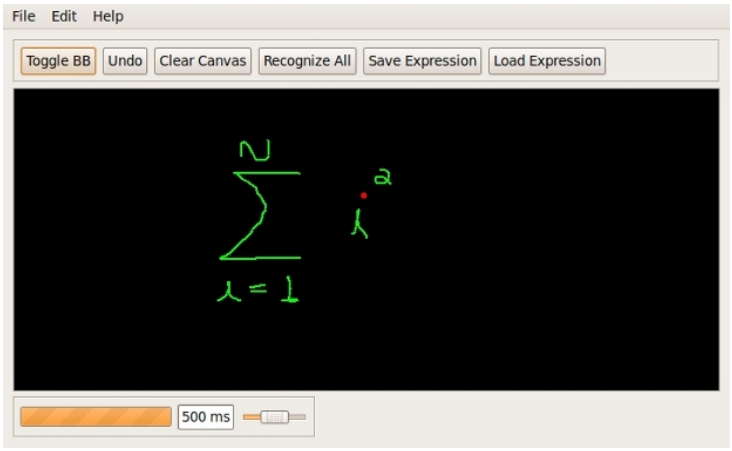
### 4.4 Estrutura do software

O ExpressMath foi desenvolvido de acordo com o padrão *Model View Controller* (MVC) de arquitetura de software. Essa estrutura visa minimizar o acoplamento entre a interface com o usuário e o resto do código, facilitando alterações em ambos.

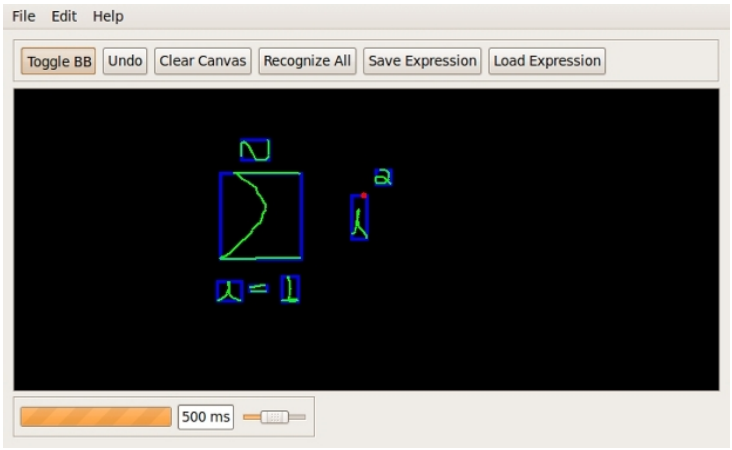
### 4.5 Melhorias Futuras

Embora nosso sistema tenha se mostrado capaz de lidar com as necessidades da maioria dos usuários, podemos listar algumas melhorias possíveis:

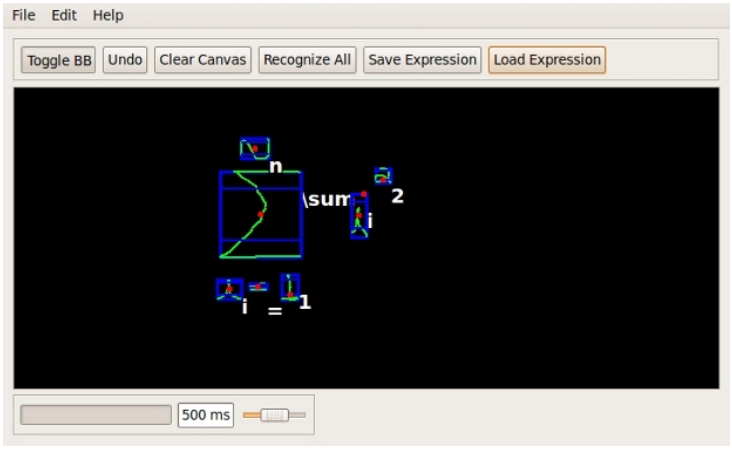
- **Interface de correção de erros:** Uma funcionalidade útil que não foi implementada ainda nessa ferramenta é o redimensionamento de símbolos. Acreditamos que, munida dessa característica, seja possível corrigir qualquer expressão pela interface.
- **Cadastro de símbolos:** Atualmente, a lista de símbolos conhecidos pelo sistema é armazenada em um arquivo `.xml`, e sua edição pelo usuário, embora possível, exige algum conhecimento técnico, se tornando proibitiva. Disponibilizar uma interface agradável para edição desse arquivo aumentaria a usabilidade do sistema.




(a)




(b)




(c)

Figura 9: (a) Expressão matemática a ser reconhecida. (b) Expressão com retângulos envoltórios. (c) Expressão reconhecida: símbolos com etiqueta, centróide e limiares.

- **Reconhecimento de expressões tabulares:** A abordagem utilizada neste trabalho não trata matrizes e outras expressões tabulares. No entanto, o sistema poderia ser acrescentado dessa funcionalidade por meio de uma extensão do código.

# 5 Avaliação de desempenho

A comprovação do funcionamento do método implementado não poderia ser verificada de outra maneira que não por testes. A seguir documentamos como ocorreram os testes de eficiência do analisador, e seus resultados.

## 5.1 Preparação

Para automatizar esse processo, implementamos a *feature* de salvar e carregar uma expressão em um arquivo no formato `.xml`. A interface gráfica possui atalhos claros para o uso dessa função, que foi implementada em conjunto com a equipe do classificador.

O arquivo `.xml` de entrada contém uma lista de símbolos com todas as informações necessárias para sua reconstrução, a saber:

- Etiqueta LaTeX
- Conjunto dos traços (com informação de tempo)
- Coordenadas do retângulo envoltório

Para que fosse possível testar todas as etapas do reconhecimento, inserimos também nesse arquivo de entrada a fórmula LaTeX final esperada, que pode então ser comparada com a obtida pelo método implementado.

Todos os testes foram automatizados usando a ferramenta para criação de testes parameterizados do JUnit 4, possibilitando que fossem rodados a qualquer momento, fato que auxiliou bastante na medição dos impactos de pequenas alterações nas constantes utilizadas no algoritmo.

## 5.2 Coleta de dados

A coleta das expressões para testes se deu em duas etapas, com objetivos relativamente diferentes. Todos as expressões foram coletadas com auxílio de um dispositivo do tipo *data tablet*, e classificadas manualmente por meio de um classificador mockado.

### 5.2.1 Testes de eficiência

Nesse primeiro grupo de testes estão incluídas apenas expressões escritas pelos desenvolvedores do sistema, e seu objetivo foi testar o comportamento de cada uma das etapas do algortimo.

As expressões desse conjunto são em geral simples, mas apresentam algumas das principais irregularidades frequentemente encontradas em expressões matemáticas manuscritas.

Esse conjunto de testes consiste de 111 expressões matemáticas, e o sistema foi capaz de reconhecê-las corretamente em 90,8% dos casos.

### 5.2.2 Testes de aceitação

Nessa etapa dos testes o objetivo era garantir que o sistema se comportava aceitavelmente em situações reais de uso. Para isso buscamos ajuda de voluntários para entrada de expressões pré-definidas. Dez voluntários participaram, sendo que cada um escreveu as seguintes expressões:

- **1:** $a^2 = b^2 + c^2$
- **2:** $2^{3+4} + 2$
- **3:** $2^{3_4} + 5_n - 2^{x_2}$
- **4:** $\sqrt[3]{x^4 - x^2 + x^{e^{21}}}$
- **5:** $S = S_0 + V_0 t + \frac{at^2}{2}$
- **6:** $x + \sum_{m=0}^{6} y^{\frac{a+b}{m}}$
- **7:** $x = \frac{b + \sqrt{b^2 - 4ac}}{2a}$
- **8:** $\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{x^2}{\sigma^2}} dx$

O aproveitamento do sistema nesse teste não foi tão alto, em virtude das variações na escrita de alguns usuários. Das 88 expressões capturadas, apenas 62 foram reconhecidas corretamente na primeira tentativa, representando 70,5% do universo de testes.

Nas expressões erroneamente analisadas, utilizamos a interface de correção de erros, contando o número de movimentos necessários para que o ExpressMath acertasse a análise estrutural. Nesse contexto, foi possível corrigir 70% das fórmulas com apenas uma movimentação. As 30% restantes foram corrigidas em menos de quatro movimentos.

Apesar das inconsistências na escrita dos diferentes voluntários, foi possível, por meio da interface *drag & drop*, corrigir todas as expressões analisadas incorretamente.

Essa etapa dos testes foi muito importante como *feedback* no desenvolvimento, e mostrou claramente a importância e eficiência da interface para correção de erros. Em não poucos casos de teste, voluntários escreveram expressões de formação duvidosa, ou até incorreta, forçando os limites do algoritmo.

### 5.2.3 Análise dos resultados

Os testes realizados durante o desenvolvimento comprovaram a eficiência do método estudado, que se mostrou robusto no tratamento de expressões de variados níveis de complexidade. Avaliações realizadas até a presente data mostram que os resultados concordam com os apresentados no trabalho que serviu de referência [11].

A baixa taxa de reconhecimento apresentada pelo sistema nos testes com voluntários ressaltou a importância da interface de correção de erros. Sem essa funcionalidade, a usabilidade do sistema seria afetada, uma vez que além da escrita manuscrita ser muito variável, o uso de uma *data tablet* pode prejudicá-la.

## 6 Considerações finais

Descrevemos um método conhecido para análise estrutural de expressões matemáticas manuscritas fortemente baseado no conceito de dominância entre símbolos e operadores e que faz uso de *baselines* e árvores geradoras mínimas. Descrevemos também um sistema computacional que implementamos para a análise estrutural de expressões matemáticas manuscritas, baseado no método descrito. Nossa implementação gera ao final da análise estrutural a representação da expressão em LaTeX, um dos formatos mais utilizados na criação e edição de notação matemática.

O método implementado é um dos mais recentes encontrado na literatura sobre esse tema. Segundo os autores dessa abordagem, ela ainda apresenta deficiências no que diz respeito à eficiência, robustez e reconhecimento de estruturas complexas como, por exemplo, matrizes e sistemas de equações. Essas mesmas deficiências foram verificadas nesse projeto. Algumas expressões mais complexas como as que incluem estruturas tabulares, como matrizes, definições de funções e operadores com limites em pilha, não são corretamente reconhecidas.

Devido à vastidão dos símbolos possíveis numa expressão matemática, é muito difícil obter um algoritmo que saiba lidar com todos. Nesse projeto consideramos o tratamento de um conjunto de símbolos condizente com aqueles encontrados na maioria das expressões e fórmulas na literatura científica. Esse conjunto pode ser incrementado com novos símbolos, tarefa que listamos como melhoria futura.

As implementações desenvolvidas serão úteis para melhor avaliar essa abordagem baseada na construção de MST's e consequentemente para identificar as suas deficiências. Isto será importante para guiar os próximos passos nessa linha de pesquisa.

## Referências

[1] D. Blostein and A. Grbavec, *Recognition of mathematical notation*, Handbook of character recognition and document image analysis (H. Bunke and P. Wang, eds.), World Scientific,

1997, pp. 557 – 582.

[2] K. F. Chan and D. Y. Yeung, *Mathematical expression recognition: A survey*, International Journal on Document Analysis and Recognition **3** (2000), 3 – 15.

[3] S. Chang, *A method for the structural analysis of two-dimensional mathematical expressions*, Information Sciences **2** (1970), 253 – 272.

[4] R. Y. L. Chow, P. H. S. Oliveira, and E. G. C. Pires, *SisTREO – sistema titanium de reconhecimento de escrita online*, `http://www.linux.ime.usp.br/~cef/mac499-06/monografias/pedro/`, 2006, Trabalho de formatura supervisionado.

[5] Ana Paula Santos de Mello, Eduardo Yutaca Komatsu, Fábio Marcos Eiji Okuda, and Leonardo Ka Wah Hing, *Math picasso - segmentação e reconhecimento de caracteres em expressões matemáticas manuscritas*, 2007, `http://www.linux.ime.usp.br/~eiji/mac499/index.php`.

[6] C. Faure and Z. X. Wang, *Automatic perception of the structure of handwritten mathematical expressions*, Computer processing of handwriting (R. Plamondon and C. Leedham, eds.), World Scientific, 1990, pp. 337 – 361.

[7] U. Garain and B. B. Chaudhuri, *Recognition of online handwritten mathematical expressions*, IEEE Trans Syst., Man, and Cybernetics Part B: Cybernetics **34** (2004), no. 6, 2366 – 2376.

[8] Nina S. T Hirata, *ExpressMath - reconhecimento de expressões matemáticas*, `http://www.vision.ime.usp.br/~nina/projetos/expressmath/`, 2008.

[9] Nicholas E. Matsakis, *Recognition of handwritten mathematical expressions*, Master's thesis, Massachusetts Institute of Technology, 1999.

[10] R. C. Prim, *Shortest connection networks and some generalizations*, Bell System Tech. J. **36** (1957), 1389–1401.

[11] E. Tapia and R. Rojas, *Recognition of on-line handwritten mathematical expressions using a minimal spanning tree construction and symbol dominance*, J.Llados and Y. B.Kwon **12** (2004), no. 8, 329 – 340.

[12] C. C. Tappert, C. Y. Suen, and T. Wakahara, *The state of the art in online handwriting recognition*, IEEE Transactions on Pattern Analysis and Machine Intelligence (1990), 787 – 808.

[13] H. M. Twaakyondo and M. Okamoto, *Structure analysis and recognition of mathematical expressions*, Proceedings of the third international conference on document analysis and recognition, vol. 1, 1995, pp. 430 – 437.

[14] H. J. Winkler and M. Lang, *Symbol segmentation and recognition for understanding handwritten mathematical expressions*, Progress in handwriting recognition (A. Downton and S. Impedovo, eds.), World Scientific, 1997, pp. 407 – 412.

[15] R. Zanibbi, D. Blostein, and J. R. Cordy, *Recognizing mathematical expressions using tree transformation*, IEEE Transactions on Pattern Analysis and Machine Intelligence **24** (2002), no. 11, 1 – 13.