Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science

Bachelor thesis

Multimodal mathematical formula editing for web services

submitted by

Alberto González Palomo

submitted

23.12.2010

Supervisor

Prof. Dr. Jörg H. Siekmann

Advisor

George Goguadze

Reviewers

Prof. Dr. Jörg H. Siekmann
Priv.-Doz. Dr. habil. Christoph Igel

Statement

Hereby I confirm that this thesis is my own work and that I have documented all sources used.

Saarbrücken, 23.12.2010

Alberto González Palomo

Declaration of consent

Herewith I agree that my thesis will be made available through the library of the Computer Science Department.

Saarbrücken, 23.12.2010

Alberto González Palomo

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

Mathematical formula editing remains a laborious and frustrating task which requires extensive familiarization with the exact formula editor provided with a web application.

Direct manipulation formula editors are on most cases based on an internal model of the formula that dictates how the user can modify it [PS04]. The exact steps depend on the internal model of the formula which is not visible to the user. This means that it is possible to get stuck into a state where a seemingly obvious modification of the formula requires specific steps to be done. This has been reported among others in [PWY07] (page 4) and mentioned by our colleages in the ActiveMath[1] project, where during evaluation studies in schools, students have been observed to get stuck while building fractions and having to scrape the whole expression and start over.

A multimodal user interface combines "different input or output channels in order to make using a computer more efficient, easier or both"[Rai99], and in his thesis we explore what can be done within the limitations of web browsers.

1.1 Motivation: cutting the Gordian Knot

Specially in learning or assessment situations, the user is already using all his cognitive resources for the task at hand, so it is imperative to reduce the cognitive load required by the formula input component to the absolute minimum.

To free the user from the constraints imposed by structured formula editors, we apply techniques developed for mathematical formula recognition, analyzing the expression layout after the user is done positioning symbols. It allows

[1] `http://www.activemath.org`, an eLearning environment for Mathematics developed at the University of Saarland `http://www.uni-saarland.de` and the German Research Center for Artificial Intelligence (DFKI) `http://www.dfki.de`

them to follow any path and do any corrections they want without having to take into account the internal structure of the formula. In our editor the formula is re-interpreted after each change, providing visual and (if desired) auditory feedback.

Another aspect of our approach to give users more control over their interaction with the computer is the use of multiple interaction modalities, and we investigate how they can be made available to web applications through the web browser. New input modalities like multi-touch screens that were prohibitively expensive until just a couple of years ago are now becoming mainstream.

1.2 Desired features

There are four main features that define what we want to achieve.

No current formula editor has them all yet, and we classify those editors in § 2.1 with respect to each of these features.

- Integrated in web page: formula elements are part of the document tree so that symbols are rendered using the same fonts and styles as the rest of the page.

  Different typographical styles make symbols look entirely different which is specially problematic when working with mathematical formulas. Formula editors that are put in the web page as black boxes like Java applets include their own fonts which ofter differ from the ones available to the web browser. It is possible for a symbol to be displayable by one an not the other.

- Standalone: the formula editor can work disconnected from the network.

  This is important for use in schools, as they often have unreliable and overcrowded network access, mediated in many cases through web proxy servers or other restrictions.

  Having it working locally also makes possible immediate feedback to the user about how the system understands the input. It is quite frustrating to submit an answer for a Mathematics exercise and after waiting for the server round-trip getting a “syntax error” or worse, a bad mark on the exercise because of input error.

  Still, in contrast with locally installed applications, web applications do not need installation and are always updated without any user intervention.

- Direct manipulation: the displayed formula can be modified directly, not just through actions on other parts of the user interface.

Additionally, we want *unconstrained* manipulation of the formula elements which includes having to deal with meaningless intermediate states.

- Semantic: the editor provides the meaning of the formula encoded in some unambiguous format like OpenMath[2] or Content MathML[3].

  Most formula editors deal only with the *presentation* of the formula, that is, what it looks like.

  This is enough if all we want is to show it to humans for reading, but for intelligent processing of those formulas we need to derive their meaning (or *content*) at some point. Doing it as close to the initial creator as possible enables early corrections when the computer inevitably makes some mistake trying to infer the meaning from the presentation.

1.3 Scope

In this thesis we implement a formula editor capable of handling polynomials, fractions, and mixed numbers.

It can be easily integrated in web pages and accessed with normal web browsers without additional installation or configuration, and is designed to be extended later to other mathematical domains.

Mathematical formula recognition comprises two stages [CY00]:

- Symbol recognition phase: handwritten symbols are recognized by grouping strokes.
- Structural analysis phase: the spatial relationships between those recognized symbols are analyzed to derive the expression structure.

In this work, we skip the first part by starting with symbols inserted using a keyboard, a symbol palette embedded in the web page, or external input methods provided by the platform.

[2] `http://www.openmath.org`
[3] `http://www.w3.org/TR/MathML/chapter4.html`

CHAPTER 2

Literature review

2.1 Existing formula editors

Table 2.1 shows how some representative formula editors perform according to the features we defined in § 1.2.

	Integrated in web page	Standalone	Direct manipulation	Semantic
Xpress `http://eqn.xero.ca/`	Yes (SVG)	No (layout analysis in server)	Yes	No
Wiris `http://www.wiris.com/`	No (Java)	Yes	Yes (constrained)	Yes (OpenMath)
BrEdiMa `http://zarame.com/bredima/`	Yes (HTML)	Yes	Yes (constrained)	No
MathEdit `http://wme.cs.kent.edu/mathedit/`	Yes	Yes	Yes (constrained)	Yes (CMML & OpenMath)
MathDox `http://mathdox.org/formulaeditor/`	No (Canvas)	Yes	Yes (constrained)	Yes (OpenMath)
SENTIDO® `http://matracas.org/sentido/`	Yes (MathML)	Yes	No	Yes (OpenMath)

Table 2.1: Some representative formula editors

Other formula editors either fit in one of those categories or lack more than one of the required features.

- Like Wiris: DragMath

- Like BreDIMa: FireMath (MathML instead of HTML, no keyboard input, only palettes even for numbers)
- Not integrated in web page, direct manipulation, not semantic: FFES `http://research.cs.queensu.ca/drl/ffes/` (we base our algorithm on theirs as described in Chapter 3)
- Not integrated in web page, constrained direct manipulation, semantic: Maple™(since version 10[1]), Mathematica®(since version 3.0[2]), MathType
- Integrated in web page, no direct manipulation, not semantic: ASCIIMathML
- Not integrated in web page, constrained direct manipulation, not semantic: TeXmacs, Microsoft™Word (since version 2007)
- Not integrated in web page, no direct manipulation, not semantic: OpenOffice.org, AbiWord, sitmo Equation Editor[3]

We want to combines the unconstrained direct manipulation of FFES [ZBC02] and XPress [PWY07] with the semantic encoding of formulas of SENTIDO® [GP06], implemented entirely on the web browser to avoid the need for server round trips and provide immediate feedback to the user.

2.2 Layout analysis

This is a part of the whole handwritten formula recognition pipeline. We implement an algorithm based on the DRACULAE [ZBC02] component of the Freehand Formula Entry System, FFES.

We studied several approaches and looked in detail at the following:

- Tree transformations, FFES in [ZBC02].
- Minimum spanning tree, MathFor in [TR03]. This combines the left-to-right reading of symbols from DRACULAE with the minimum spanning tree from [MV99].
- Projection profile cutting in [RRSS06]. Extends the common Optical Character Recognition (OCR) technique of projection profile cutting to mathematical expressions and identifies symbol containment which is commonly a weak area of PPC.

[1] `http://www.maplesoft.com/support/help/Maple/view.aspx?path=worksheet%2fdocumenting%2f2Dmath`

[2] `http://reference.wolfram.com/legacy/v3/RefGuide/InputandOutput.html`

[3] `http://www.sitmo.com/latex/`

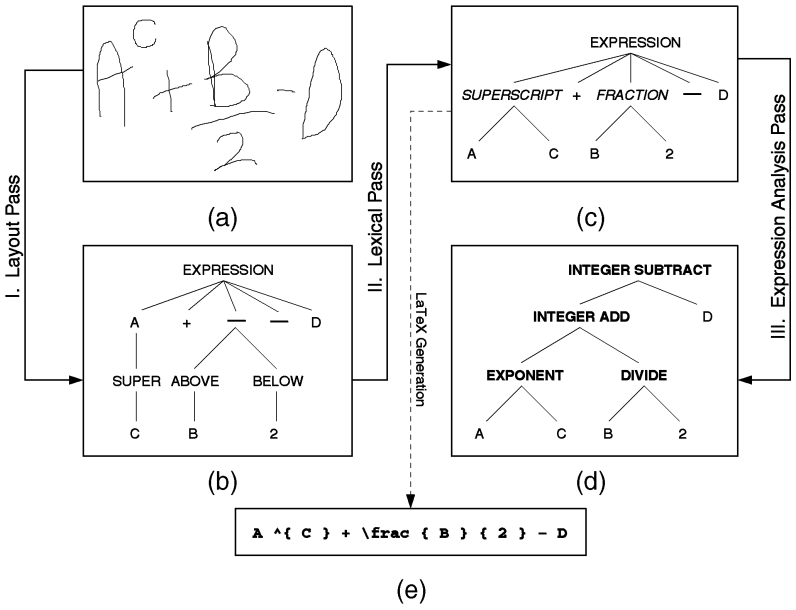


Figure 2.1: DRACULAE overview from [ZBC02]

Figure 2.2: Minimum spanning tree examples from [MV99]

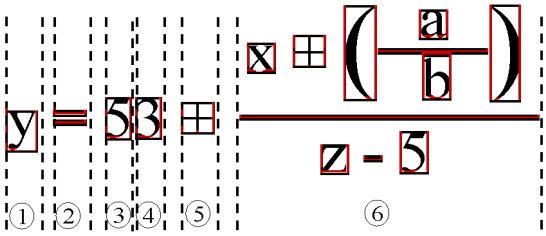


Figure 2.3: First step in profile cutting, from [RRSS06]

The decision to use DRACULAE as starting point is that we had seen a modified version of the algorithm working well in XPress (although it runs on the server and there is no publicly available source code) and that the whole source code of DRACULAE was available so we could inspect it to clarify details that might not be mentioned in the paper. That made us confident that we could get concrete results within the time limits for a Bachelor's Thesis.

CHAPTER 3

Layout analysis

In this implementation we focus on two kinds of spatial relationships: vertical alignment in the same line of text for exponents, and vertical stacking for fractions.

We implemented a variant of the DRACULAE algorithm from [ZBC02] (page 5, “Layout Pass”) to build a Baseline Structure Tree, which describes the expresion’s layout.

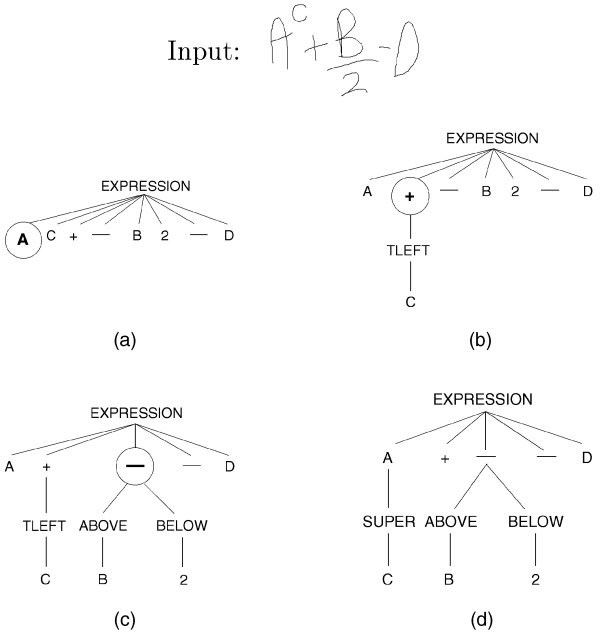


Figure 3.1: BST construction in DRACULAE, from [ZBC02]

Their implementation is based on tree rewriting, expressed as declarative transformation rules that are then applied by the TXL language interpreter. TXL is a purely functional tree rewriting language. We have reformulated the BST computation part in terms more adequate for an imperative language like JavaScript.

For instance, the way regions are attached to a symbol in DRACULAE is by determining the position of each new symbol relative to the base symbol with Partition, and appending a new region node with the corresponding label to it. Later, another pass is done with MergeRegions for each region label to combine the contents of all regions with that label into a single region.

However, since the labels are pre-determined we can do it all in one pass, creating a new slot in the base symbol object for each new region label and inserting subordinate symbols directly into the final slot.

Another difference is that we attach the nodes inside parenthesis to the `CONTAINS` region of the opening bracket, to simplify the semantic analysis phase. We describe those details in the next section.

3.1 Computing the Baseline Structure Tree

The BST is built top-down, creating first a region node that will contain the whole expression, and inserting the symbols one by one.

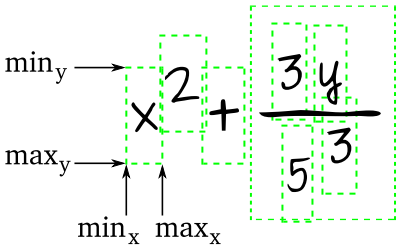


Figure 3.2: Bounding boxes of individual symbols

The bounding boxes of symbols in a formula are shown in Figure 3.2. Please note that the box around the fraction corresponds to the scaled font glyph used for the fraction bar (the “em dash”: “—”), not the whole fraction. There is no structure yet.

First, symbols are sorted from left to right using their min_x coordinate. Proceeding in that order exploits the left-to-right reading order in mathematical notation to reduce the search space for symbol relationships.

Then we create a new root expression node, and insert in it a new BST symbol node for each symbol.

The node insertion algorithm is described in Algorithm 1. Its implementation is the function `BSTNode.insert()` in page 111.

Data: BSTNode objects $tree$ and $symbol$
Result: $symbol$ is inserted in $tree$ either as a direct child or attached to a previous symbol there

if `IsRegion(`*tree*`)` **then**
  try to insert the symbol into each symbol in this region, from right to left, stop at first success
  if the symbol was not attached to any child, append it here
else
  pos := `Position(`*tree, symbol*`)`
  if $pos \neq NULL$ **then**
    `Insert(`*tree.children[pos], symbol*`)`
  end
end

Algorithm 1: BSTNode.insert(node)

The function `Position` is implemented as `BSTNode.determine_position()` in page 110 and computes the region label for the new node relative to the base one. Those region labels are shown in Figure 3.3. We use the same names as DRACULAE.

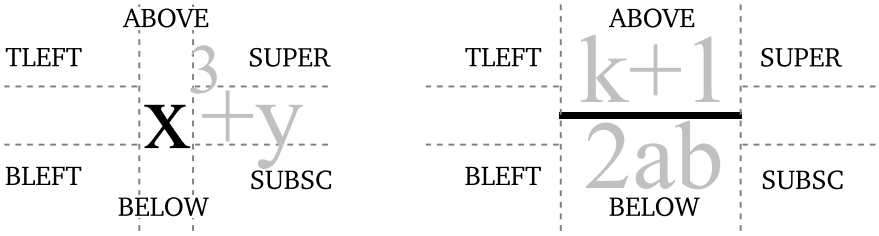


Figure 3.3: Regions around a symbol where subordinate symbols are attached

A NULL value for the position means that either the tree is empty, or the new node is in the same baseline as the rightmost symbol in the first level of the tree.

A concrete example

The following is the sequence of steps for recognizing the formula in Figure 3.2:

$$x^2 + \frac{3y}{5^3}$$

After sorting the nodes from left to right according to their min_x position we create a new region node $expression$ and proceed to insert the symbols {x, 2, +, —, 3, 5, y, 3} into it in order.

1. Insert x. The region is empty, so it just gets appended to it.

2. Insert 2. We look at the symbols in $expression$, from last inserted to first, and find x. The position of 2 wrt. x is SUPER, and the node 2 is inserted at x.SUPER.

3. Insert +. Again we compute the position of + wrt. x, which in this case is NULL since it does not belong to any of them. Therefore it gets appended to $expression$, that now contains {[x]{ SUPER([2]) }, +}

4. Insert —. After computing its position wrt. + and x, it does not belong to any of their regions and it gets appended to $expression$.

5. Insert 3. Looking at its last child node, the fraction bar, its position is ABOVE, so it gets inserted there.

6. Insert 5. Like 3, only this time its position wrt. the fraction bar is ABOVE. At this point the fraction bar subtree is [—]{ ABOVE([3]) BELOW([5]) }

7. Insert y. Recursing into the tree, it ends up in the ABOVE region of the fraction bar, after 3.

8. Insert 2. Going down the tree we reach the BELOW region of the fraction bar. There we look at its children from right to left, and find that 2 belongs in the SUPER position of 5, so we insert it there: [5]{SUPER([2])}

9. The final tree is then EXPRESSION([x]{ SUPER([2]) }, [+], [—]{ ABOVE([3], [y]) BELOW([5]{ SUPER([2]) }) })

In the scope of this thesis, all subordinate symbols (exponents, subscripts, numerator and denominator) start right of the main symbol. Therefore finding the dominant symbol means just taking the leftmost one.

However in other cases like the summatory in Figure 3.4, the subordinate symbols (the limits of the summatory) can start left of the main one.

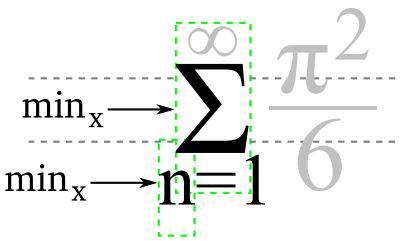


Figure 3.4: The summation limits are subordinated to the Σ symbol but start at its left

Subordination of symbols is defined in pages 6-7 of [ZBC02] in the functions Start, Contains and Overlaps. We have collected their definition of symbol dominance in a more compact symbolic form in Figure 3.5.

$$
\begin{aligned}
a \text{ dominates } b := \;& a \text{ overlaps } b \\
& \vee\ a \text{ contains } b \\
& \vee\ a.symbol\_class = \text{Variable range} \wedge b\ \neg\text{adjacent}\ a \\
a \text{ overlaps } b := \;& a.symbol\_class = \text{Nonscripted} \\
& \wedge\ a.min_x \leq b.centroid_x < a.max_x \\
& \wedge\ \neg(b \supset a) \\
& \wedge\ \neg(b.symbol\_class = \text{Open}|\text{Close Bracket} \\
& \qquad \wedge\ b.min_y \leq a.centroid_y < b.max_y \\
& \qquad \wedge\ b.min_x \leq a.min_x) \\
& \wedge\ \neg(b.symbol\_class = \text{Non scripted}|\text{Variable range} \\
& \qquad \wedge\ b.max_x - b.min_x > a.max_x - a.min_x) \\
a \supset b := \;& a.symbol\_class = \text{Root} \\
& \wedge\ a.min_x \leq b.centroid_x < a.max_x \\
& \wedge\ a.min_y \leq b.centroid_y < a.max_y \\
a \text{ adjacent } b := \;& a \text{ and } b \text{ belong to the same base line}
\end{aligned}
$$

Figure 3.5: Formal definition of the symbol dominance relation

As mentioned above we do not need it yet, but it will be necessary for the planned future extension of our implementation.

Another difference is related to Table 1 in page 4 of their paper, which contains the symbol classes. We do not need the different y-centroids and thresholds because the DOM nodes are already normalized as seen in Figure 3.6.

In the fraction numerator, the 3 and the y in $3y$ appear to be in different base lines because of their different graphical shape. That is what DRACULAE has to correct. In our case, what we receive from the web page Document Object Model is the bounding boxes of the font glyphs whose height, by definition, is constant for a given font size. There are however disadvantages too, like the big bounding box for the fraction bar that forces us to take it into account when determining the regions for the symbols above and below it.

3.2 Semantic analysis and translation

Obtaining the layout tree from the symbol positions is just the first part.

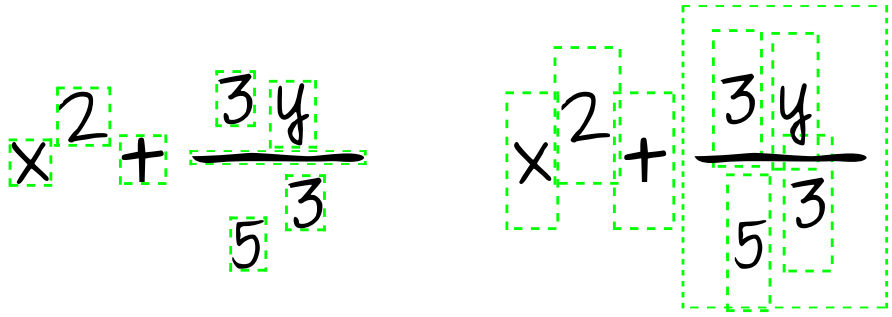


Figure 3.6: Graphical bounding boxes (left) versus font glyph bounding boxes (right)

In order to obtain the meaning of the formula, we need to do what in DRACULAE is called "semantic analysis". This involves parsing the symbols and regions in a given baseline to resolve their grouping according to operator precedence and associativity.

One difference is that in our algorithm parentheses grouping is resolved in the layout analysis phase. For that we extend DRACULAE's symbol classes with two new ones, `CLOSE_BRACKET` and `BRACKETED`, and use the `CONTAINS` region, which in DRACULAE is only for square roots, for the contents between the parentheses.

Our function `determine_position()` returns `CONTAINS` when the new symbol is added to the right of an `OPEN_BRACKET` node, and its centroid is within the min_y and max_y bounds of the bracket.

When the new symbol is a `CLOSE_BRACKET`, we add it and change the symbol class of the opening parenthesis to `BRACKETED` to indicate that it is a complete bracketed object. This way we recognize nested parentheses correctly.

The advantage of doing this grouping at this point is that it simplifies later translations. The layout analysis functions already have the information needed for it, so it seems wasteful to throw it away and force later processors to do the work again. However as we note next this would not be necessary if producing a semantic encoding was not a requirement.

A great majority of the existing formula editors reviewed in § 2.1 deal only with the presentation of the formula. That simplifies things notably as many details can be just passed over to the final recipient of the result formula encoding, be it a human or an interpreter like LaTeX, a MathML renderer, or a Computer Algebra System.

For instance, let us consider a simple expression like $(x + 1)^2$.

Without the parenthesis grouping we just described the BST would look like in Figure 3.7.

The exponent 2 should not be attached to the closing parenthesis, but to the whole expression. However, if we produce some string output from it, it just so happens that we can get away with it. For instance, if the output format

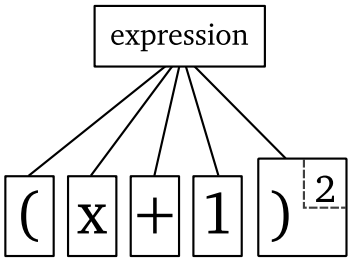


Figure 3.7: A basic structure tree that suffices for some applications

is LaTeX, we would write each of the characters "(", "x", "+", "1", then reach the last node and output ")^2" for it. The result would be "(x+1)^2" which is correct, even if the formula was never correctly interpreted in the first place.

If we want to process the meaning of the formula, even if it is just to write it in some unambiguous semantic encoding format like OpenMath, we must "go the extra mile" and resolve those details.

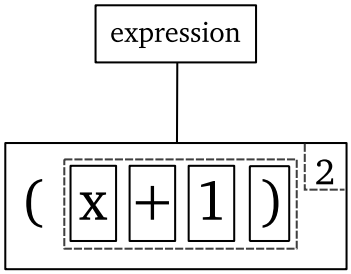


Figure 3.8: The BST produced by the layout analysis in TACTO

The same applies to operator precedence and associativity which we discuss in the next section. It is possible to edit a formula without dealing with it and then encoding the result in plain strings, but then we are just passing the problem of interpreting the formula to its recipient.

Parsing the layout tree: precedence and associativity

To resolve the precedence and associativity of operators in a same baseline, we implement a simple shift-reduce parser[1]. We just need a small contex-free grammar, because some of the operations like the fraction bars, exponentiation and parenthesized grouping were already resolved in the layout analysis phase.

The remaining productions are in Figure 3.9.

[1] http://en.wikipedia.org/wiki/Bottom-up_parsing#Shift-reduce_parsers

$$
\begin{aligned}
S &\rightarrow\ S\,! \\
S &\rightarrow\ S \times S \\
S &\rightarrow\ S + S \\
S &\rightarrow\ S - S \\
S &\rightarrow\ S = S \\
S &\rightarrow\ S < S \\
S &\rightarrow\ S > S \\
S &\rightarrow\ \text{NUMBER} \\
S &\rightarrow\ \text{VARIABLE} \\
S &\rightarrow\ \text{SYMBOL}
\end{aligned}
$$

Figure 3.9: Simple grammar for the remaining parsing after computing the BST

It is implemented in the function `parse_operators()` in page 126.

Composite symbols

To recognize composite symbols we do a pass through the BST and combine them into single nodes, for which we define three additional node types: `BSTNode.INTEGER`, `BSTNode.REAL`, and `BSTNode.FUNCTION`[2].

Doing this process in the BST allows us to take spacing into account, which we need to distinguish whether the decimal separator (point, comma, etc.[3]) is really part of a number or something else. If the separation between it and the next digit is bigger than the width of one digit, then the separator is not part of a decimal number.

This is implemented in the function `combine_symbols()` in page 119.

3.3 Translation to standard formats

We have implemented three translators so far:

MathML : we can produce this directly from the BST without any semantic analysis because the precedence of operators does not matter: if parentheses are needed to apply an operator before a higher-precedence one, they would be already in the BST as the user would have written them.

[2]Function composite symbols (e.g. “sin”, “cos”) are not yet recognized but it will be necessary in future extensions.

[3]`http://en.wikipedia.org/wiki/Decimal_mark`

OpenMath : the shift-reduce parser we mentioned above is applied to each region to resolve the operator application in each baseline. Then the generated subtree is translated recursively into XML nodes. Superscripts are translated into applications of the `arith1:power` operator, and subscripts as applications of `algebra1:vector_selector` so that x_1 means the first item of the vector x.

Plain English : we generate this from the OpenMath encoding. The implementation is in page 91. It can produce text also for some OpenMath symbols that are not yet supported by the formula editor, like the n-th root `arith1:root`.

CHAPTER 4

User interface

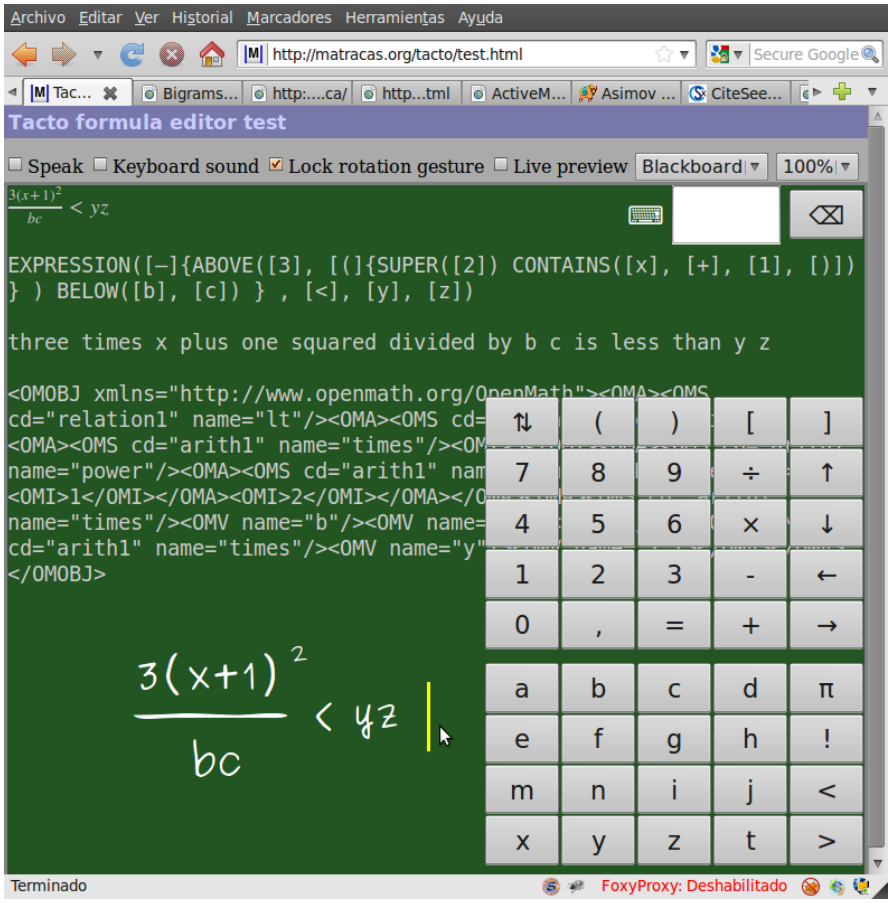


Figure 4.1: General view of the test application included in this thesis

4.1 Direct manipulation

We allow the user to manipulate elements directly and without any contraints. Any element can be moved independently of the rest.

One advantage versus structured editors is that we can insert first all the symbols we want to use without caring about their relative positions, and move them afterwards. This is particularly relevant for input modalities that do not include a keyboard, and some handwriting recognition input methods cover the screen which prevents normal interaction with the GUI beneath it.

In TACTO we can, for instance, type $x2 + 3/ab$, exit the input method window and then rearrange the symbols, moving the 2 up, the fraction bar under $x^2 + 3$, and ab under it. A structured editor would require us to exit the input method to navigate the formula structure and get each part inserted in the right place.

It works the same way on mobile devices, as they typically have small screens that are mostly covered by the on-screen keyboards or other input methods. In Figure 4.2 we see how in our editor, the user can call up the onscreen keyboard, simply type the symbols that will be used in the formula even without seeing it, then dismiss the keyboard and have the whole screen free for rearranging them. The inserted part "-6" is then selected with the lasso gesture in the fourth picture, and finally dragged to its end position.

4.2 Touch interfaces

Browsers designed for tablet devices behave differently from their desktop implementations. In particular, some gestures we would want to use are already taken for some operations like page scrolling that are done in alternative ways in desktop browsers.

We need to account for those differences to provide a useful user interface for each platform, while keeping the usage similar enough that a user proficient with one of those variants can work with the others comfortably.

Touch interaction has been found to be faster than using a mouse [SMS09], although the error rate was bigger for small targets. We adjusted the size of the targets (buttons and palette items) in our interface accordingly. The minimum size of the buttons is 18×12 millimeters. This exceeds slightly the recommended minimum touch target sizes by various vendors in Table 4.1.

We establish an initial size in the example application Listing 1 of 10×10 mm. All user interface elements sizes inside the formula field (like the cursor and the sizing handles) are defined in terms of the font size, in "em" units, so that it suffices to specify once the font size in the document's CSS style declaration:

```
div.blackboard > * { font-size:10mm; }
```

TACTO then extracts the actual pixel size from the rendered cursor element and adapts the computations to it.

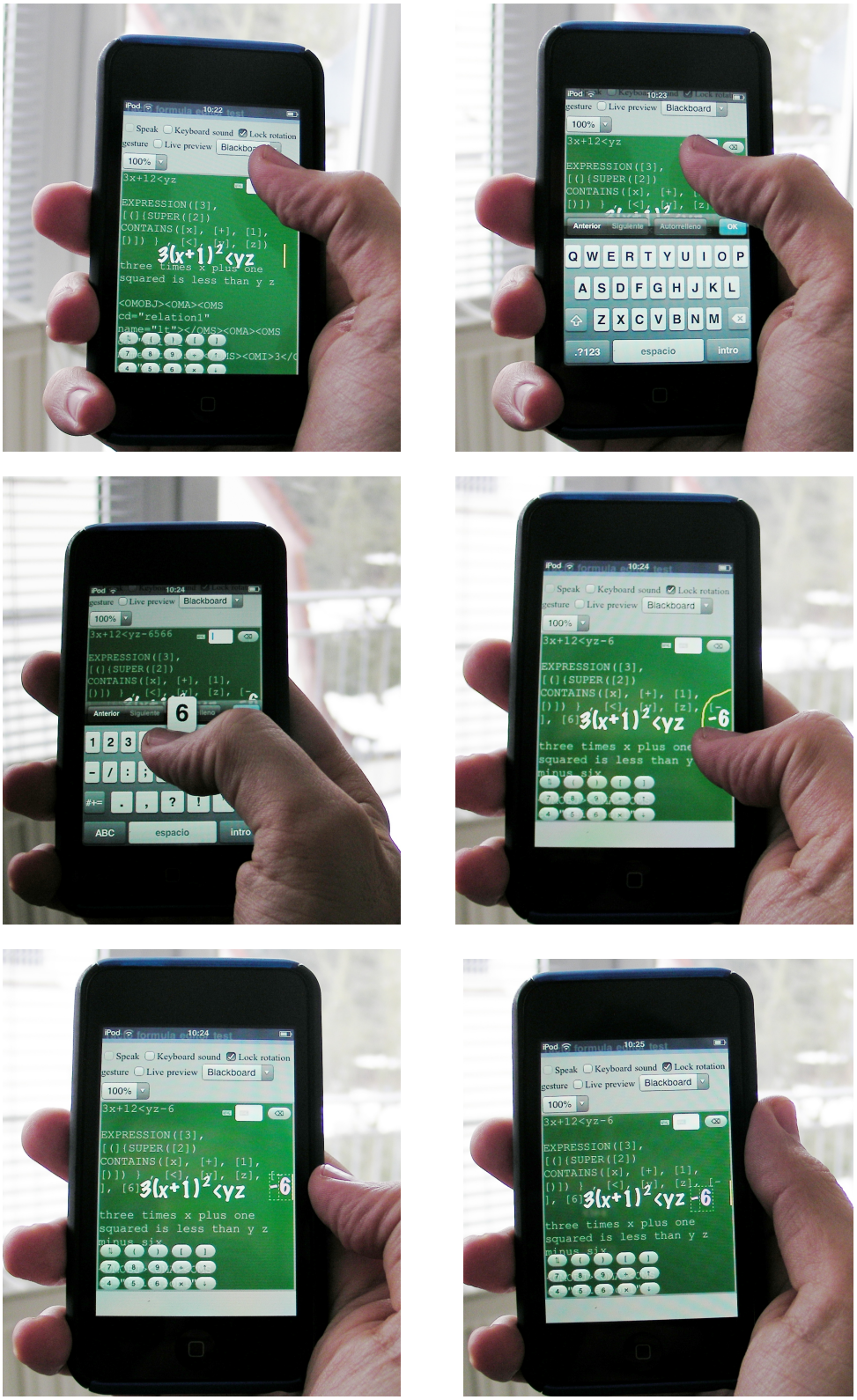


Figure 4.2: “Blind” on-screen keyboard usage on a mobile device

	Publication	Minimum size	Recommended size
Apple	iPhone Human Interface Guidelines [App10]	–	44 pixels
Microsoft	Windows Phone UI Design and Interaction Guide [Mic10]	9mm + 2mm spacing	7mm + 2mm spacing
Nokia	Designing applications for S60 5th Edition [Nok09]	10mm + 1/2mm spacing	7mm + 1mm spacing
Ubuntu	Designing for Finger UIs[Ubu10]	$10 - 14$mm	10mm

Table 4.1: Recommended touch target sizes.

4.3 Item selection and manipulation

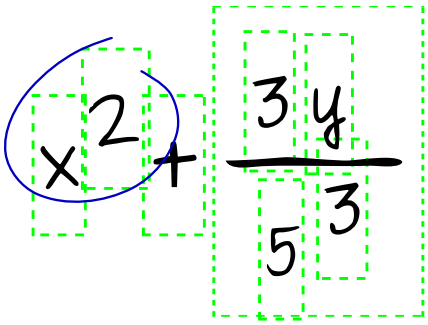
Selection

In graphics manipulation applications there are two basic types of multiple object selection: one is a rectangle that selects all items inside it, and another is known as “lasso” selection, where the user draws a curve that contains the desired objects.

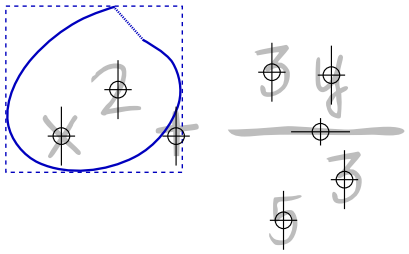
Lasso selection is more natural and precise than rectangle selection when using a touch interface. In a mouse and keyboard scenario, the Shift ⇑ key can be pressed to add items to the selection for cases where a complex selection shape is needed, but that combination is not adequate (often even not available) for touch interfaces.

The items selected are those whose centroid is inside the polygon defined by closing the given path. Determining whether a point lies inside a polygon is a common problem in Computer Graphics and there are several standard solutions. The one we use here is the even-odd rule: we trace a line from the point to somewhere outside the polygon, and count the number of intersections. If it is an odd number the point is inside, otherwise it is outside.

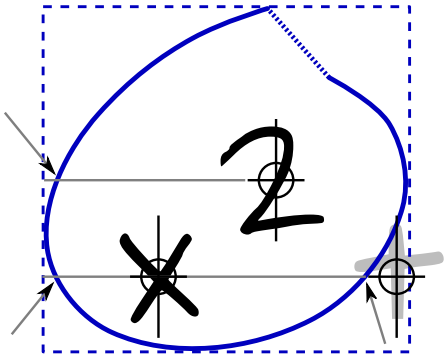
We consider only symbols whose centroids are inside the bounding box of the path, and use a horizontal line from each of them to the left of the path bounding box. This allows us to discard quickly those path segments that are completely above or under the line.



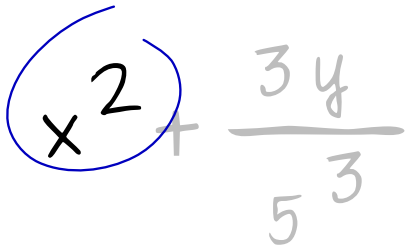
(a) Selection path, and symbol bounding boxes



(b) Selection path closed with bounding box, and symbol centroids. Only the symbols “x”, “2” and “+” are inside the box.



(c) The lines for “x” and “2” intersect the path once and are inside. The line for “+” however has two intersections.



(d) Selected symbols “x” and “2”

Figure 4.3: Selecting nodes with the even-odd rule

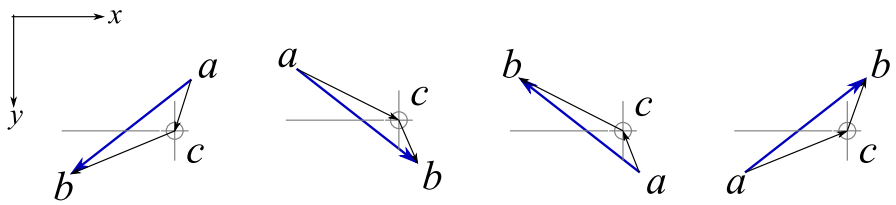


Figure 4.4: Path segments: non-degenerated cases, when $c_y \neq b_y$

There is a special case when the symbol centroid lies exactly on the same horizontal line as a point in the path: we have an intersection when c is either exactly on b, or to its right. The other cases are shown in Figure 4.4. It is not necessary to compute the intersection point, just wheter the symbol centroid is

on the right side of the path segment.

Data: centroid point c, list P of points in the selection path
Result: true if the centroid is inside the area determined by the path

if $c \in$ `BoundingBox(`P`)` **then**
  $count := 0$
  $a :=$ `Last(`P`)` ; `/* this closes the path */`
  $i := 0$
  foreach $b \in P$ **do**
    if $\vec{cb}_y = 0$ **then**
      if $c_x \geq b_x$ **then**
        `/* degenerated case` $c_y = b_y$ `*/`
        $count := count + 1$
      end
    else if $\vec{ac}_y > 0$ **then**
      if $\vec{cb}_y > 0 \ \wedge \ \vec{ac}_x / \vec{ac}_y > \vec{cb}_x / \vec{cb}_y$ **then**
        `/* first two cases in Figure 4.4 */`
        $count := count + 1$
      end
    else if $\vec{ac}_y < 0$ **then**
      if $\vec{cb}_y < 0 \ \wedge \ \vec{ac}_x / \vec{ac}_y < \vec{cb}_x / \vec{cb}_y$ **then**
        `/* last two cases in Figure 4.4 */`
        $count := count + 1$
      end
    end
    $a := b$
  end
  return $count \mod 2 = 1$
else
  return *false*
end

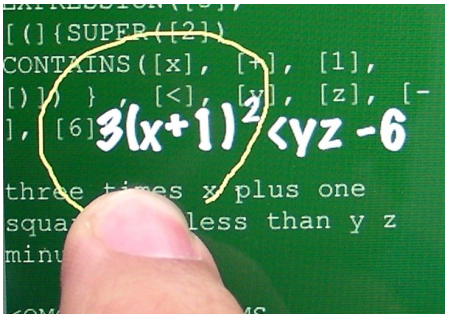
Algorithm 2: Determining whether a point is encircled by a path

To draw the curve on the screen we use a SVG path. The editor detects whether SVG support is available on the browser, and if not it uses a fallback method: the path bounding box is used for selection, shown on the screen using an HTML `<div>` element.

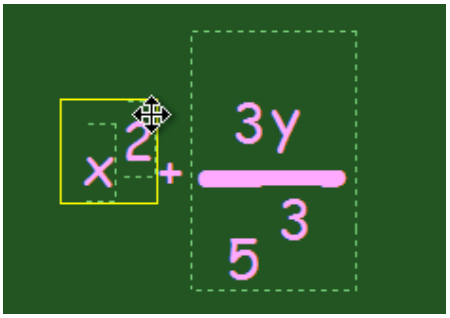
Resizing

Some symbols need to be resizeable by the user. Fraction bars are resized horizontally, and brackets are resized vertically.

The handles for resizing are set in the outside of the bounding box. This is necessary to avoid the symbol being obscured by them when their size is comparable (small symbols).

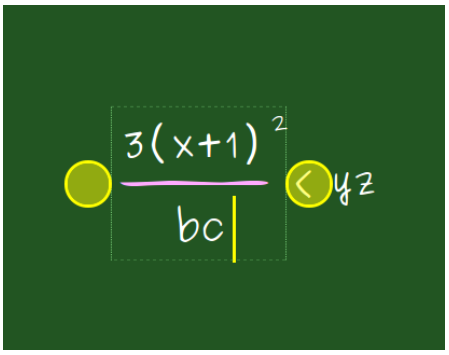


(a) Lasso selection using SVG in Safari Mobile on an iPod touch

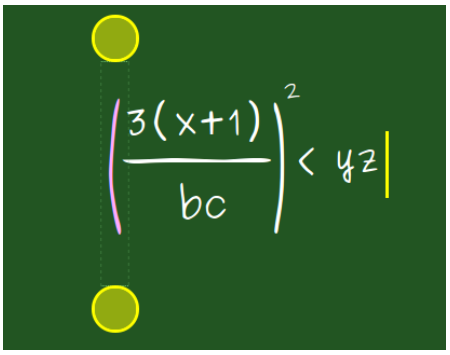


(b) Fallback HTML rectangle selection in Internet Explorer 8

Figure 4.5: If the browser does not support SVG, the editor automatically switches to a simple rectangle selection



(a) Fraction bar sizing handles



(b) Left parenthesis sizing handles

Figure 4.6: Sizing handles (yellow circles) for stretchable symbols

To avoid rounding errors we refine the fixpoint of the font size computation for a given pixel size. For a given target size, we compute a tentative font size and set it in the element. Then we measure the actual size and refine the font size factor if necessary. Finally we use the corrected factor to compute the final font size that will be used. The implementation is in the function `scale_items` in page 71.

Gesture recognition

We distinguish single-touch and multi-touch gestures.

The multi-touch gesture we implemented is known as pinch-zoom: touching the screen with two fingers, setting them apart increases the size of the formula, bringing them together decreases it, and moving both fingers simultaneously moves the formula around.

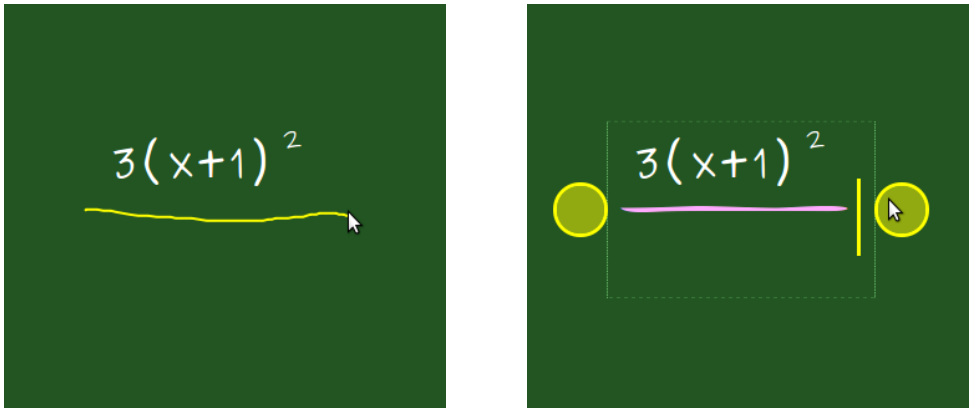


Figure 4.7: Fraction bar gesture (left), and inserted fraction bar glyph (right)

A two-finger touch gesture has four degrees of freedom: distance between fingers, 2D position, an rotation. That is more than a usual mouse[1] which has 2D position and the scroll wheel. The main advantage of the pinch zoom gesture is that it allows simultaneous adjustment of size and position. Even mapping the mouse wheel to the zoom value, moving an object around and adjusting its size at the same time would involve pressing the left mouse button while dragging the mouse and controlling the mouse wheel. That is quite ackward compared to the multi-touch gesture.

Positioning and scaling the formula works by computing the corresponding affine transformation and applying it with CSS transformations.

What we receive from the browser in the gesture event is the distance between fingers and the angle. We obtain the coordinates from the lower-level touch events that are also delivered.

After transforming the formula container element the event coordinates we receive are still in the screen coordinate system, so we need to apply the inverse transformation to them to map them to the right place in the transformed formula coordinate system. This is done in the usual way, just inverting the transformation matrix.

The single-touch gestures that are recognized so far are lasso selection and fraction bar (Figure 4.7).

We look at the general shape of the gesture path: if the overall width is greater than a minimum (we use half the font line height), the first point of the path lies near the left side, the last point near the right side, and the width is at least five times the height, it is recognized as a fraction bar gesture. Otherwise it is processed as a selection gesture. The recognizer can be extended in the function `gesture` in page 91.

[1]There exist mice that work with drawing tablets and are rotation-sensitive.

4.4 Speech synthesis

Recent developments in web standards provide some degree of audio control, enough to implement concatenative speech synthesis[2] which basically consists on playing voice sound samples in sequence to form utterances. It is an standard technique in embedded devices where processing power is limited, and that fits the web browser environment too.

The natural language generation is done from the OpenMath form, because we are interested in the meaning and not in the appearance of the expression [Ram95].

The disambiguation work is already done in the layout and semantic analysis when translating to OpenMath, so starting from that point our speech synthesis module only needs to generate English prose and combine the speech samples to read it aloud. The production of English text from the OpenMath encoding of the formula is described in § **??**.

Speech sample collection

Concatenative synthesis requires voice samples of each piece of speech. We collected them from the speech synthesizer included in MacOS X to save time: recording the same sound from a person would require first preparing adequate conditions for the recording, and then spending more time in audio post-processing. Using the output of the speech synthesizer gave us a noise-free sound source with repeatable results and normalized loudness. However, the procedure we describe can be applied to real voice samples.

For each fragment, we need two variants: one in the middle of an utterance (medial), and another at the end (final) where the change in the f_0 (fundamental frequency) curve and other phonological features like phoneme duration are most noticeable in English.

For each of the words used in our natural language generation (listed in page 98), we produced an audio file with the output of the speech synthesizer when reading each word twice, to capture the medial and final variants. Then we split each file manually using a graphical audio editing application as shown in Figure 4.8 where the fine vertical line is the point we chose to split the file. In normal speech though, there is no separation between the words as can be seen in Figure 4.9 where the two utterances of the word “hundred” get melded together.

As we see there it is not always easy to split the words so in some cases we inserted phonemes with easily recognizable shapes like /t/ in between words to ease the segmentation work.

[2] `http://en.wikipedia.org/wiki/Speech_synthesis#Concatenative_synthesis`

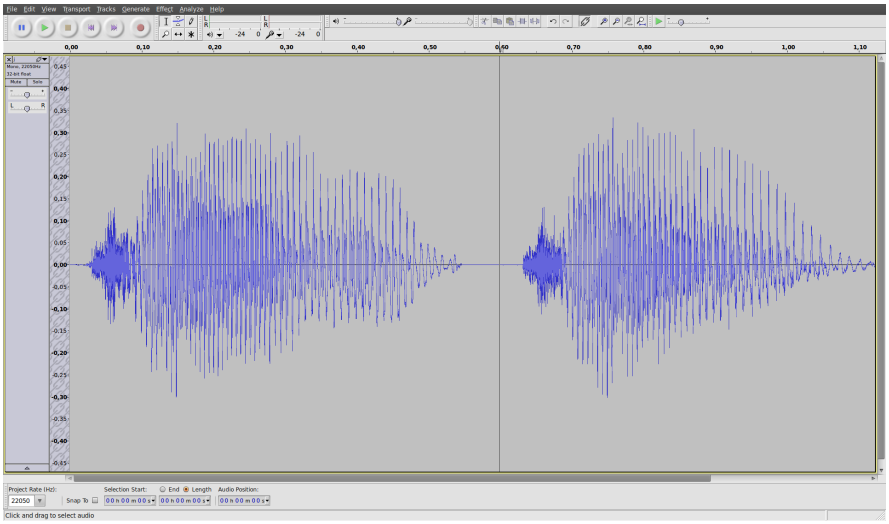


Figure 4.8: Sound wave for the name of the letter “j” repeated twice

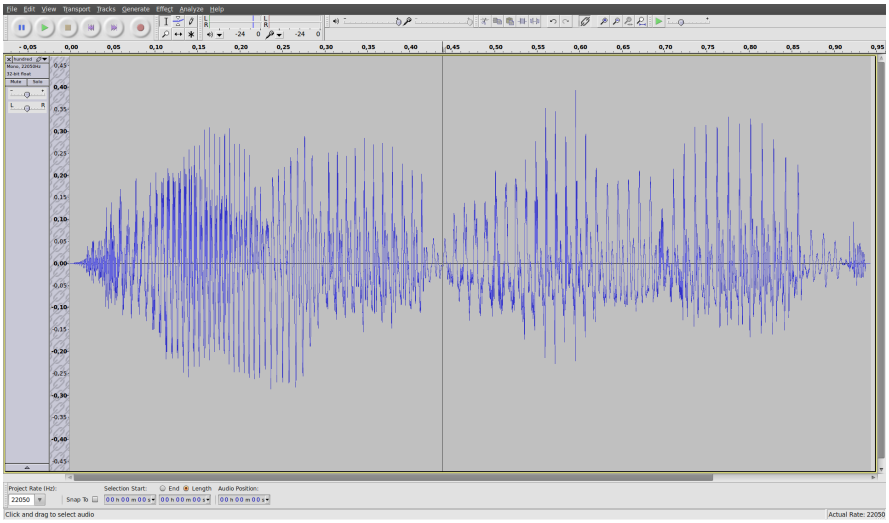


Figure 4.9: Sound wave for the word “hundred” repeated twice. This is a more typical situation in continuous speech.

Voice sample concatenation

In recent years there have been great advances in the features available to web applications. The current effort to standarize them is known collectively as HTML5, and it includes an audio interface [wha] that allows fine control over audio playing. It is supported in most current browsers[3].

[3]The exception as usual is Internet Explorer.

That functionality is exposed to JavaScript through the `Audio` object. To concatenate the samples that make up a sentence, we need to start playing the next exactly when the current finishes. The `Audio` object emits a `ended` event when the sound has played to the end. We found however that the timing was unreliable and also varied greatly from one browser to another. Firefox 3.6 had long delays which produced unacceptable pauses between words, while Chrome 8.0 delivered the events too quickly and the samples partially overlapped each other. Opera 9 was somewhat in the middle with clear delays but shorter than Firefox.

Since we have a fixed set of samples we want to play, a better approach is to store their duration and then use a timer to launch each sample at the right time, independently of the `ended` event.

Each sample is stored with its duration in milliseconds (the `voice_metadata` variable in page 98).

Using this procedure the sound output is consistently accurate in Firefox, Safari, Chrome and Opera. Safari 5 in MacOS X 10.5 needs a little delay (we use 10ms) before playing the first sample, but otherwise works fine.

What does not work well is Safari Mobile in iOS 3.2.2 which we suspect is limited to playing only one sound sample at a time. When they overlap slightly, some are not played. If we set the timer to fire after a longer duration, all samples are played one after another, but that introduces pauses in the other browsers. Also, the `ended` event is sent only once, after all samples have been played. In iOS 4.2 Apple disabled automatic sound playing in the browser, so it does not work at all unless the user disables the pop-up window blocker[4].

4.5 Use with external input methods

Alternative input methods are being incorporated into mainstream operative systems, and can be also used with TACTO if they provide keyboard event simulation. We report here a couple of possibilities.

Handwriting recognition

Nowadays there are built-in or freely available handwriting recognition input methods for all three major platforms: Linux, MacOS X and Windows.

In our preliminary tests we got acceptable results from an input method for Linux called CellWriter[5] that we show in Figure 4.10. It requires prior training and we do not always get the symbols at first try, but it left a good impression after working with it for a while.

Another one we tried was Inkwell, developed by Apple Computer for the Newton MessagePad and included later in MacOS X. This one does not need

[4]Such pop-ups are typically used for intrusive advertisement which is why they are blocked by default. We would hesitate asking our users to disable that filter.

[5]CellWriter's web page is at `http://risujin.org/cellwriter/`

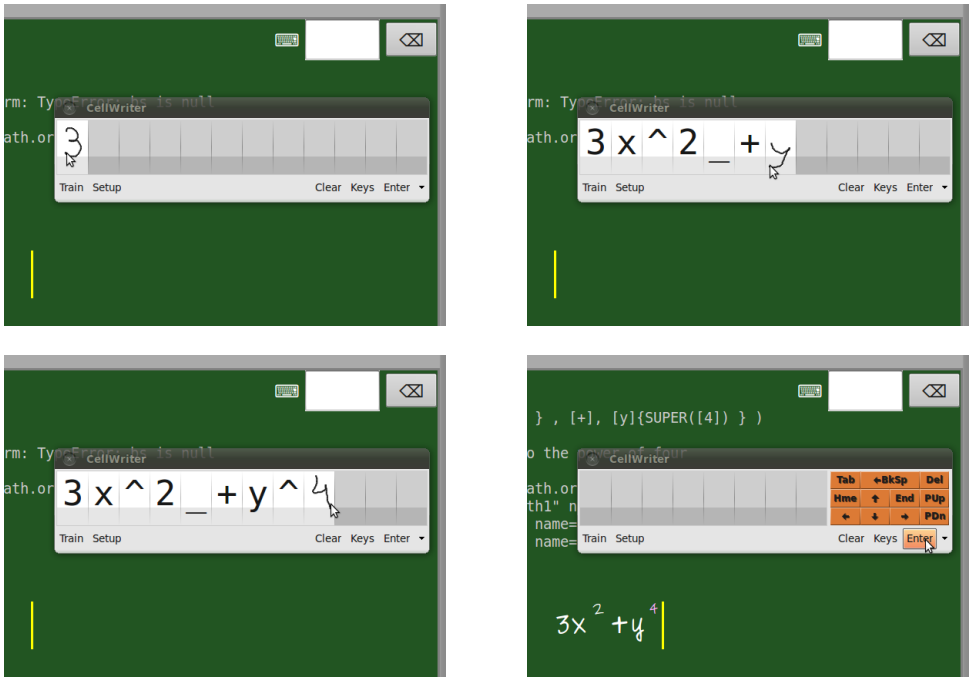


Figure 4.10: Handwriting recognition provided by external input methods through keyboard event simulation

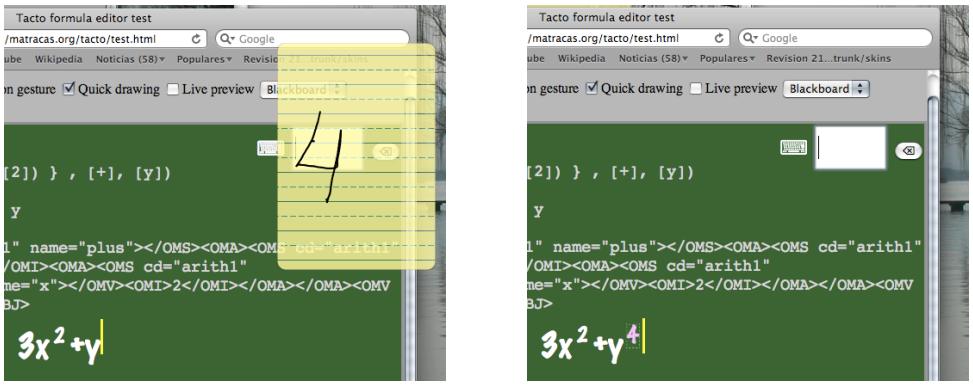


Figure 4.11: Inkwell handwriting input method in MacOS X

any training, but we found a bit more difficult to enter a few test characters. A successful instance can be see in Figure 4.11.

Voice input

Voice input is supported in the same way as handwriting recognition, via simulated keyboard events. This is a standard feature in current versions of the Windows[6] and Android[7] operative systems.

We tried in Windows and the results, even after training the recognizer, were quite disappointing. We had expected to get phonetically similar words to the spoken ones which we could correct by computing some metric like the Levenshtein[8] distance between the recognized words and the symbols and commands supported in our editor, but the recognized text was so different from the spoken words that we could see no way of correlating them.

One cause of the problem is that, from the web browser, there is no way to restrict the vocabulary so the recognizer must consider any possible utterance. We suspect that restricting the vocabulary to the symbols actually supported by our editor would enhance the results considerably. We could do that by sending the audio data to a server for recognition, the way Google implemented it in Android, but that falls outside of the scope of this thesis and is left for future experimentation.

[6] Vista/7 `http://www.microsoft.com/enable/products/windowsvista/speech.aspx`

[7] 2.2 Froyo `http://developer.android.com/resources/articles/speech-input.html`

[8] `http://en.wikipedia.org/wiki/Levenshtein_distance`

CHAPTER 5

Implementation

We have implemented our editor in "object-oriented JavaScript[1]" to keep it well organized. The whole editor is contained in an object with the name "Tacto_editor", and that is the only symbol in the global name space. That eliminates name clashing problems.

Another consequence is that the editor is re-entrant, that is, there can be as many instances of the formula editor in a single page as desired, and the program code is loaded only once per page.

5.1 Integration in an existing web application

Given an empty `<div>` element where we want the formulas to be edited, let's say it was assigned to a variable named `blackboard`, we give it to the constructor:

```
var tacto = new Tacto_editor(blackboard);
```

That `<div>` element must have a height set by the embedding application: otherwise, being empty, the browser would collapse it into a horizontal line. In our example it's done with CSS as folows:

```
div.blackboard { height:40em; }
```

The formula editor constructor takes care of everything else.

[1] `http://www.crockford.com/javascript/private.html`

Pre-populating the formula content

In this version we provide some rudimentary support for setting the formula content before the user starts modifying it. In principle each symbol must be inserted by moving the cursor to any location and inserting text:

```
tacto.move_to(40, 260);
tacto.insert("x^2_+3");
```

To simplify it a bit, the insert function accepts a mini-language for layout, where "^" means move cursor up and "_" move cursor down. Other characters are inserted verbatim.

Symbol palettes

Additional user interface elements like button palettes can be built in HTML as usual, and that is the responsability of the embedding application. In our example application in Listing 1 we add a button palette for devices without a physical keyboard and a text input field for invoking the on-screen keyboard or other input methods provided by the operative system.

The embedding application can request being notified when the formula content changes by setting a callback function in the `tacto.onchange` variable. That function will be called with one argument, the `Tacto` object, so that the function can distinguish between several formula editors being used simultaneously.

5.2 Cross-browser compatibility

Our pupose is to support a variety of web browsers, at least those that follow industry standards and for the rest provide some basic functionality in those most widely used.

The editor works correctly in the following HTML engines and browsers:

- Gecko: Firefox 3.x, Fennec (Firefox mobile)
- WebKit: Google Chrome, Safari, Safari Mobile (iPod, iPhone, iPad)
- Trident: Internet Explorer 8
- Presto: Opera 10

Text editing operations

Our first tests used the `contenteditable` HTML attribute that activates the built-in rich text editor in desktop browsers (Firefox, Explorer, Safari, Opera). However, according to Apple's Technical Note TN2262[2] "contenteditable is not supported in Safari on iPhone OS.". As the iPad and iPhone are some of our target platforms, we had to use another approach.

It is possible to handle the key events (key press, key down, key up) and insert the text ourselves in the document. This requires also drawing our own caret, the cursor that shows the insertion point for text.

One important drawback of this solution is the different handling of text input methods in different platforms. For instance, the standard Spanish keyboard has "dead keys" for the accents: pressing one of the keys for the accents does not insert anything, but signals that the next letter will have to be composed with the accent to yield the actual character that will be inserted. Firefox on Mac OS X however does not report the dead key events, so we get the unaccented letter which in incorrect. In contrast with German, there is no correct way to replace the accented letters (called "umlaute" in German) with sequences of unaccented letters. The same problem happens with any input method that maps key sequences to characters, like those used for Chinese.

Another option is to use a normal HTML text input field. In our tests, this allowed to enter text using input methods provided by the operative system that did not work otherwise, like Pinyin for Chinese or handwriting recognition (§ 4.5). To activate this option the user selects the "keyboard" text input field, situated on the top right in our test application.

Any changes in the content of the input field are replicated into the main area like those characters were input in any other way. We show a couple of examples with handwriting recognition in § 4.5 and with an on-screen keyboard in Figure 4.2.

Symbol resizing

Some symbols must be able to stretch, like fraction bars and parentheses.

Our first approach was to use SVG as it allows affine transformations on elements. However it did not work as expected because HTML content inside the SVG elements was not displayed.

Finally, using HTML elements and CSS did work to our satisfaction. Sizing the symbols is done using the font size. As there is no uniform relationship between font size and actual size of the symbol's bounding box (it varies among browsers and operative systems) we compute a linear factor when the formula editor first receives focus.

[2] `http://developer.apple.com/safari/library/technotes/tn2010/tn2262/index.html`

Initially, the `<div>` element for the cursor has a single character inside, a vertical bar, so that it is sized according to the font size specified with CSS. Once we have the font size factor, we remove that content to leave the cursor element empty with just the border that indicates the text insertion point.

For fractions bars, stretching them horizontally in this way makes them thicker instead of only lengthening them. To keep a uniform thickness regardless of the size, we tried using CSS borders. This works, but the lines do not match the stroke style in any fonts where dashes are not entirely straight lines (like the callygraphic fonts we use in most screenshots), and the line thickness also looks out of place and can not be automatically adjusted. Apart from that, the color for the borders must be set explicitly in addition to the text color.

The best solution was to scale the font as needed, and then reduce the line thickness by applying an affine transformation to it. This requires a modern browser with support for CSS 3 transformations (Firefox, Chrome, Safari, Opera) and it is ignored otherwise.

Exponents and subscripts are scaled automatically. Our first versions allowed manual scaling of any symbol, but in our limited user testing it became clear that it was not effective.

Instead we found a much better solution which is to change the symbol sizes during the layout analysis. This turns out to be fast enough in small formulas to be done continuously in the background, giving the user immediate feedback about how the computer understands the formula.

In Figure 5.1 we see what happens when the user drags the symbol 2 around the others. As their relative position changes, those understood as super- or sub-indexes reduce their size. First the 2 is an exponent, but by lowering it the user converts it into a factor of x. Lowering it still makes it a sub-index of x, and finally bringing it to the bottom left corner of the expression makes the whole $x + 1$ part to become its exponent.

By default, this automatic resizing only takes place when the user finishes a symbol drag gesture. We chose to be conservative in this respect and not assume that the editor is running on a fast computer. In the test application there is a checkbox in the upper options row labelled “Live preview” which activates that feature. A timer is used to limit the updates to be at least 100 milliseconds apart, in an attempt to keep CPU usage withing reasonable bounds. Whenever the user stops moving the mouse/finger, after 100ms the screen contents will be updated.

5.3 Performance: latency

Latency is a major problem which is often incorrectly dismissed as reported in [MW93]:

> Lag has been shown to degrade human performance in motor-sensory tasks on interactive systems. At 75 ms lag, the effect is

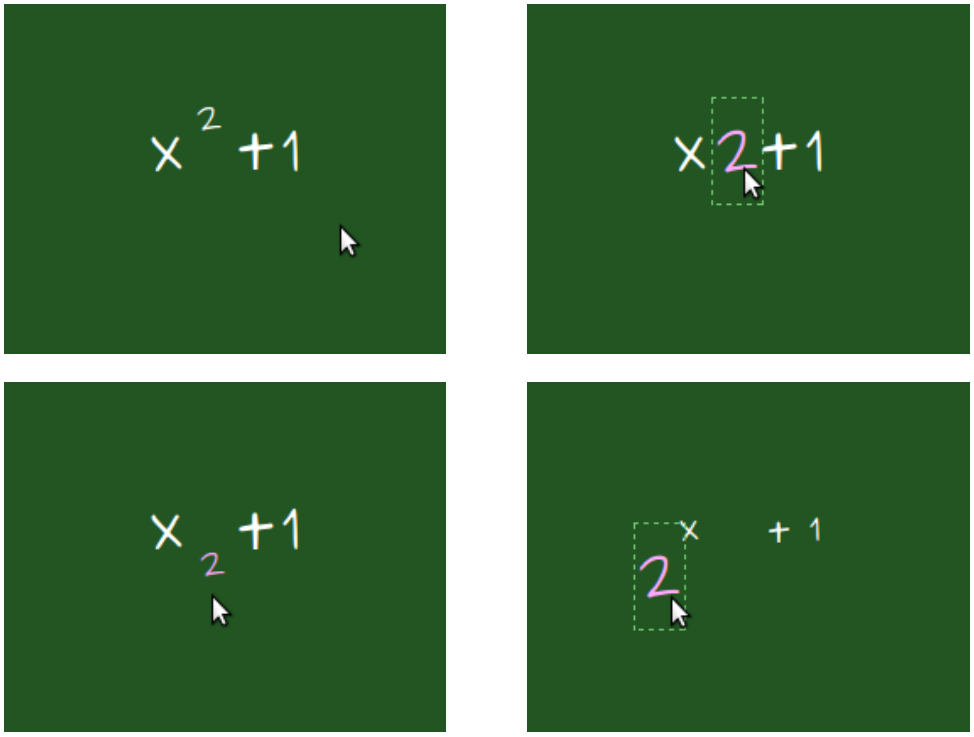


Figure 5.1: As the user drags the "2" around, the sizes of the symbols change to show how TACTO interprets them

> easily measured, and at 225 ms performance is degraded substantially. A model has been presented showing a strong multiplicative effect between lag and Fitts' index of difficulty. The model explains 93.5% of the variation in observations.

Therefore a responsive user interface is not merely a desirable feature, but a fundamental one. We have devoted a considerable amount of effort to find ways of reducing the latency.

The obvious way is careful analysis of the execution path and removal of repeated calculations, which we did.

However there are sources of lag that do not depend on the program's efficiency or computing power available.

In touch screen devices that allow finger gestures, the system can not tell immediately whether a finger touching the screen is the beginning of a click operation or of a more complex gesture. This causes a noticeable delay in reporting click events. However those touch events are also reported, so we consider the finger touching the screen as a complete click event thus eliminating the disambiguation latency.

5.4 Application Programming Interface

These are the properties and methods publicly available in the `Tacto_editor` object.

move_to(x**,** y**)**
Move the cursor to the given coordinates.

move_by($delta_x$**,** $delta_y$**)**
Increase the cursor position with the given coordinate deltas.

insert($content$**)**
Insert the content into the formula at the current cursor position. If the content is an enumerable object (like a character string), each of its elements is inserted in sequence.

do_command($command$**)**
Execute the given command.

up : move cursor up

down : move cursor down

left : move cursor left

right : move cursor right

backtab : move left eight spaces

tab : move right eight spaces

delete : delete forward

backspace : delete backwards

get_expression($formats$**)**
Update the Baseline Structure Tree if necessary, and derive from it the formula encodings in the given formats. The formats supported are:

pmml : Presentation MathML, produced directly from the BST.

openmath : OpenMath semantic encoding.

bst : the BST as is.

text : natural language form of the formula, built from the OpenMath form. We only implemented a limited English grammar but it could be extended, also to other languages.

voice : speech synthesis from the natural language form. The returned value is a `Voice` object (page 97) with methods `speak()` and `stop()`.

options
This object holds several flags accesible to the embedding web application.

lock_rotation : the rotation dimension of the multi-touch gesture is ignored. This is the default as rotating the formula does not seem to have many practical uses apart from confusing the user and/or doing fancy demos.

live_preview : the formula is processed as the user moves its elements, without waiting for the current gesture to finish.

show_cursor($value$**)**
Show the cursor if $value = \mathrm{true}$, or hide it otherwise. Sometimes we want to show the cursor even after the formula loses the keyboard focus, for instance when we manipulate it from outside like the symbol palettes and the keyboard input text field do.

font_changed
Used by the embedding application to tell TACTO that the font changed so it should refresh any related computations. The font size is used among other things for symbol stretching and it must be kept accurate.

attach_text_input
Attaches an external text input field so that TACTO gets automatically notified when its contents change. This is how input methods provided by the operative system are applied.

CHAPTER 6

Conclusion

6.1 Summary

We have implemented a mathematical formula editor for mathematical formulas including polynomials, fractions and mixed numbers, with the features we identified in § 1.2.

The editor supports keyboard input, mouse or single touch operations and gestures, and multi-touch gestures. It uses both graphical and speech output.

It works in all major web browsers and platforms, with graceful degradation when some features like SVG are not provided.

6.2 Future work

We plan to extend the TACTO editor to support other mathematical domains. The earlier new symbols we plan to implement are the square root and integrals.

The square root is more difficult to implement than others becase it must stretch in two dimensions. If we used SVG to paint it ourselves it would not be a problem but then it would not match the document fonts, which as we mentioned in the introduction is an important requirement for us.

The integrals however are quite simple. The integral sign stretches vertically just like the parenthesis and we have tested that succesfully. It can indeed be handled like an open parenthesis, with the closing one being the “dx”, and this last detail is the only missing part. We suspect it would not be difficult but it is outside of the scope of the thesis so we decided to use that time for other aspects.

Another interesting feature for future development is importing of formulas in different formats, not just exporting as we have now. This can be done with a formula layout algorithm. A detailed description of the TEX algorithm can

be found in [HW97], although a simpler one might suffice at first. Another option would be to use an existing library for that like jsMath `http://www.math.union.edu/~dpvc/jsMath/` which at first sight it seems to builds up the formula in HTML in a very similar to the input of our algorithm.

Bibliography

[App10] Apple Inc., *iPhone Human Interface Guidelines*, 2010, available online at http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/DesigningNativeApp/DesigningNativeApp.html#//apple_ref/doc/uid/TP40006556-CH4-SW16.

[CY00] Kam-Fai Chan and Dit-Yan Yeung, *Mathematical expression recognition: a survey*, International Journal on Document Analysis and Recognition **3** (2000), no. 1, 3–15, available online at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.55.952.

[GP06] Alberto González Palomo, *Sentido: an authoring environment for OMDoc*, OMDOC – An open markup format for mathematical documents [Version 1.2], LNAI, no. 4180, Springer Verlag, August 2006, available online at http://omdoc.org/pubs/omdoc1.2.pdf.

[HW97] Reinhold Heckmann and Reinhard Wilhelm, *A functional description of tex's formula layout*, J. Funct. Program. **7** (1997), 451–485, available online at http://www.cs.uni-sb.de/~heckmann/papers/neuform.ps.gz.

[Mic10] Microsoft Corp., *Windows Phone UI Design and Interaction Guide Version 2.0*, 2010, available online at http://go.microsoft.com/?linkid=9713252.

[MV99] Nicholas E. Matsakis and Paul A. Viola, *Recognition of handwritten mathematical expressions*, Master's thesis, 1999, available online at http://www.research.microsoft.com/~viola/Pubs/DocExtract/matsakis99recognition.pdf.

[MW93] I. Scott MacKenzie and Colin Ware, *Lag as a determinant of human performance in interactive systems*, INTERCHI (Stacey Ashlund, Kevin Mullet, Austin Henderson, Erik Hollnagel, and Ted N. White, eds.), ACM, 1993, available online at http://www.yorku.ca/mack/CHI93b.html, pp. 488–493.

[Nok09] Nokia, *Designing applications for S60 5th Edition*, 2009, available online at http://library.forum.nokia.com/index.jsp?topic=/S60_5th_Edition_Cpp_Developers_Library/GUID-5486EFD3-4660-4C19-A007-286DE48F6EEF.html.

[PS04] Luca Padovani and Riccardo Solmi, *An investigation on the dynamics of direct manipulation editors for mathematics*, In Third International Conference on Mathematical Knowledge Mnagement, LNCS 3119, Springer, 2004, available online at http://www.di.unito.it/~padovani/Papers/lncs_3119.pdf.

[PWY07] Marco Pollanen, Thomas Wisniewski, and Xiao Yu, *Xpress: A novice interface for the real-time communication of mathematical expressions*, Electronic Proceedings of the Workshop on Mathematical User Interfaces (MathUI), 2007, available online at http://www.activemath.org/workshops/MathUI/07/proceedings/Pollanen-xpress-MathUI07.pdf.

[Rai99] Roope Raisamo, *Multimodal human-computer interaction: a constructive and empirical study*, 1999, available online at http://acta.uta.fi/pdf/951-44-4702-6.pdf.

[Ram95] T. V. Raman, *Aster - towards modality-independent electronic documents*, Proceedings of the 1995 DAGS Conference on Electronic Publishing and the Information Superhighway (Boston, Massachusetts), May 1995, available online at http://www.cs.cornell.edu/Info/People/raman/publications/dags-95.ps, pp. 59–71.

[RRSS06] Amar Raja, Matthew Rayner, Alan P. Sexton, and Volker Sorge, *Towards a parser for mathematical formula recognition*, Mathematical Knowledge Management (MKM) (Wokingham, UK) (J. M. Borwein and W. M. Farmer, eds.), Lecture Notes in Artificial Intelligence (LNAI), vol. 4108, Springer Verlag, Berlin, Germany, August 2006, available online at http://www.cs.bham.ac.uk/~aps/papers/RaRaSeSo-MKM06-ParserMFR.pdf, pp. 139–151.

[SMS09] F. Sasangohar, I. S MacKenzie, and S. D Scott, *Evaluation of mouse and touch input for a tabletop display using fitts' reciprocal tapping task*, Human Factors and Ergonomics Society Annual Meeting Proceedings, vol. 53, 2009, available online at http://www.yorku.ca/mack/CHI93b.html, p. 839–843.

[TR03] Ernesto Tapia and Raúl Rojas, *Recognition of on-line handwritten mathematical expressions using a minimum spanning tree construction and symbol dominance*, GREC (Josep Lladós and Young-Bin Kwon, eds.), Lecture Notes in Computer Science, vol. 3088, Springer, 2003,

available online at http://www.inf.fu-berlin.de/inst/ag-ki/ger/grec03.pdf, pp. 329–340.

[Ubu10] Ubuntu, *Designing for Finger UIs*, 2010, available online at https://help.ubuntu.com/community/UMEGuide/DesigningForFingerUIs.

[wha] *Html5 (including next generation additions still in development), draft standard — 10 december 2010*, available online at http://www.whatwg.org/specs/web-apps/current-work/multipage/video.html#audio.

[ZBC02] Richard Zanibbi, Dorothea Blostein, and James R. Cordy, *Recognizing mathematical expressions using tree transformation*, IEEE Trans. Pattern Anal. Mach. Intell. **24** (2002), no. 11, 1455–1467, available online at http://www.cs.queensu.ca/home/cordy/Papers/TPAMI_Math.pdf.

Source code

The source code for the TACTO formula editor is available at the author's web page: `http://matracas.org/tacto/`

There are three files. The first is an example web page that embeds the editor, the second is the support JavaScript for the example, and the third is the implementation of the formula editor proper.

Example HTML page

Figure 1: An example HTML file showing how to embed the TACTO editor in an HTML page

```
<!DOCTYPE html>
<!-- Copyright 2010 Alberto González Palomo http://matracas.org/
     Author: Alberto González Palomo

     English:
       This Program may be used by anyone in accordance with
       the terms of the German Free Software License.

       The License may be obtained under
       http://www.d-fsl.org

     Deutsch:
       Dieses Programm kann durch jedermann gemäß den Bestimmungen
       der Deutschen Freien Software Lizenz genutzt werden.

       Die Lizenz kann unter http://www.d-fsl.de
       abgerufen werden.
     -->
```

```
<html manifest="cache.manifest">
  <head>
    <meta http-equiv="Content-Type"
          content="text/html;charset=utf-8" />
    <meta name="apple-mobile-web-app-capable" content="yes" />
    <meta name="viewport"
          content="user-scalable=no, width=device-width" />
    <meta name="apple-mobile-web-app-status-bar-style"
          content="black-translucent" />
    <link rel="apple-touch-icon" href="apple-touch-icon.png" />
    <title>Tacto formula editor test</title>
    <link rel="stylesheet" href="test.css" type="text/css" />
  </head>
  <body onload="init()">
    <div id="header">
      <h1>Tacto formula editor test</h1>
      <div class="controls" id="controls">
        <input type="checkbox"
               name="voice-active" id="voice-active"
               onchange="display_formula()"
               /><label for="rotation-locked">Speak</label>
        <input type="checkbox" name="sound-active" id="sound-active"
               onchange="keyboard_click_active = this.checked"
               /><label for="sound-active">Keyboard sound</label>
        <input type="checkbox"
               name="lock_rotation" id="lock_rotation"
               onchange="tacto.options[this.name] = this.checked"
               checked="checked"
               /><label
                    for="lock_rotation">Lock rotation gesture</label>
        <input type="checkbox"
               name="live_preview" id="live_preview"
               onchange="tacto.options[this.name] = this.checked"
               /><label for="live_preview">Live preview</label>
        <select title="Graphic style"
                onchange="set_theme(this.value)">
          <option value="blackboard">Blackboard</option>
          <option value="paper">Paper</option>
        </select>
        <select title="Window zoom"
                onchange='document.body.style.zoom = this.value'>
```

```
        <option value="0.5">50%</option>
        <option value="1.0" selected="selected">100%</option>
        <option value="1.5">150%</option>
      </select>
    </div>
  </div>
  <div id="editor" ontouchmove="event.preventDefault()">
    <div id="blackboard" class="blackboard">
      <div onmousedown="palette_click(event)"
           ontouchstart="palette_click(event)"
           style="position:absolute; top:0; right:0; z-index:4">
        <div class="palette" style="display:block"
             id="palette-mini">
          <label for="keyboard-input">⌨</label>
          <input name="keyboard-input" size="4"
                 id="keyboard-input"
                 onfocus="tacto.show_cursor(true)"
                 autocapitalize="off" autocorrect="off"
                 placeholder="⌨" />
          <input type="button" value="⌫" name="backspace" />
        </div>
      </div>
      <div id="formula-display"></div>
      <div onmousedown="palette_click(event)" id="palettes"
           ontouchstart="palette_click(event)"
           onclick="blackboard.focus()"
           style="position:absolute; bottom:0; right:0; z-index:4;
                  -webkit-transition:height 0.5s ease-in-out;
                  -moz-transition:height 0.5s ease-in-out;
                  -o-transition:height 0.5s ease-in-out;">
        <div class="palette">
          <table>
            <tr>
              <td><input type="button" value="↑↓"
                         title="Hide/show the palette"
                         name="toggle-palettes" /></td>
              <td><input type="button" value="(" /></td>
              <td><input type="button" value=")" /></td>
              <td><input type="button" value="[" /></td>
```

```
        <td><input type="button" value="]" /></td>
      </tr>
      <tr>
        <td><input type="button" value="7" /></td>
        <td><input type="button" value="8" /></td>
        <td><input type="button" value="9" /></td>
        <td><input type="button" value="÷" /></td>
        <td><input type="button" value="↑"
                   name="up" /></td>
      </tr>
      <tr>
        <td><input type="button" value="4" /></td>
        <td><input type="button" value="5" /></td>
        <td><input type="button" value="6" /></td>
        <td><input type="button" value="×" /></td>
        <td><input type="button" value="↓"
                   name="down" /></td>
      </tr>
      <tr>
        <td><input type="button" value="1" /></td>
        <td><input type="button" value="2" /></td>
        <td><input type="button" value="3" /></td>
        <td><input type="button" value="-" /></td>
        <td><input type="button" value="←"
                   name="left" /></td>
      </tr>
      <tr>
        <td><input type="button" value="0" /></td>
        <td><input type="button" value="," /></td>
        <td><input type="button" value="=" /></td>
        <td><input type="button" value="+" /></td>
        <td><input type="button" value="→"
                   name="right" /></td>
      </tr>
    </table>
  </div>
  <div class="palette">
    <table>
      <tr>
        <td><input type="button" value="a" /></td>
        <td><input type="button" value="b" /></td>
        <td><input type="button" value="c" /></td>
        <td><input type="button" value="d" /></td>
```

```
              <td><input type="button" value="π" /></td>
            </tr>
            <tr>
              <td><input type="button" value="e" /></td>
              <td><input type="button" value="f" /></td>
              <td><input type="button" value="g" /></td>
              <td><input type="button" value="h" /></td>
              <td><input type="button" value="!" /></td>
            </tr>
            <tr>
              <td><input type="button" value="m" /></td>
              <td><input type="button" value="n" /></td>
              <td><input type="button" value="i" /></td>
              <td><input type="button" value="j" /></td>
              <td><input type="button" value="&lt;" /></td>
            </tr>
            <tr>
              <td><input type="button" value="x" /></td>
              <td><input type="button" value="y" /></td>
              <td><input type="button" value="z" /></td>
              <td><input type="button" value="t" /></td>
              <td><input type="button" value="&gt;" /></td>
            </tr>
          </table>
        </div>
      </div>
    </div>
  </div>
  <div id="message"
       style="overflow:scroll; height:16em; display:none"></div>
  <audio id="keyboard-click" src="pip.wav" type="audio/wav"></audio>
  <script type="text/javascript" charset="utf-8"
          src="tacto.js"></script>
  <script type="text/javascript" charset="utf-8"
          src="test.js"></script>
 </body>
</html>
```

JavaScript functions for the example HTML page

Figure 2: JavaScript functions for the example page

```
var blackboard = document.getElementById( "blackboard" );
var tacto;// Constructed later in init()

function $(id) { return document.getElementById(id); }

function message(text)
{
    var container = $( "message" );
    if (container.firstChild)
    {
        container.insertBefore(document.createElement( "br" ),
                               container.firstChild);
        container.insertBefore(document.createTextNode(text),
                               container.firstChild);
    }
    else
    {
        container.appendChild(document.createTextNode(text));
    }
}

// Keyboard sound:
var keyboard_click_active = false;
var keyboard_click = $( "keyboard-click" );
if (keyboard_click_active)
{
    keyboard_click.load();
    keyboard_click.autobuffer = true;
}

var display_timer;
function palette_click(event)
{
    tacto.standard_event(event);
    var prevent_default = true;
    if (event.target.nodeName.toLowerCase() == "input" )
    {
```

```
if (keyboard_click_active)
{
    (new Audio(keyboard_click.src)).play();
}
switch (event.target.name)
{
case "up":
case "left":
case "down":
case "right":
case "backspace":
case "baseline":
    event.preventDefault();
    tacto.do_command(event.target.name);
    blackboard.focus();
    break;
case "keyboard":
    event.preventDefault();
    $("keyboard-input").focus();
    break;
case "keyboard-input":
    break;
case "get-tree":
    event.preventDefault();
    display_formula();
    blackboard.focus();
    break;
case "toggle-palettes":
    event.preventDefault();
    var node = $("palettes");
    if (node.style.height == "auto" || !node.style.height)
    {
        node.style.height = "2em";
    }
    else
    {
        node.style.height = "auto";
    };
    blackboard.focus();
    break;
default:
    event.preventDefault();
    tacto.insert(event.target.value);
```

```
            blackboard.focus();
        }
    }
    tacto.show_cursor(true);
}

function display_formula_error(message)
{
    var error = document.createElement("div");
    // This is for Internet Explorer:
    if ("textContent" in error) error.textContent = message;
    else                        error.innerText   = message;

    return error;
}
var previous_voice;
function display_formula()
{
    var formula_display = $("formula-display");
    while (formula_display.firstChild)
        formula_display.removeChild(formula_display.firstChild);
    var div = document.createElement("div");
    var formats = ["pmml",
                   "bst",
                   "openmath",
                   "text"];
    if ($("voice-active").checked) formats.push("voice");

    var expression = tacto.get_expression(formats);
    var result, i;
    result = expression.pmml;
    if (typeof result == "string")
    {
        div.appendChild(display_formula_error("MathML: " + result));
    }
    else
    {
        for (i = 0; i < result.length; ++i) div.appendChild(result[i]);
    }
    div.appendChild(document.createElement("br"));
    div.appendChild(document.createElement("br"));
    result = expression.bst;
    if (typeof result == "string")
```

```
{
    div.appendChild(display_formula_error("BST: " + result));
}
else
{
    div.appendChild(document.createTextNode(result));
}
div.appendChild(document.createElement("br"));
div.appendChild(document.createElement("br"));
result = expression.openmath;
if (typeof result == "string")
{
    div.appendChild(display_formula_error("OpenMath: " + result));
}
else
{
    var serializer = new XMLSerializer();
    var sentences = expression.text;
    if (typeof sentences == "string")
    {
        sentences = ["Error when building English form: "
                     + sentences];
    }
    for (i = 0; i < result.length; ++i)
    {
        if (sentences && i < sentences.length)
        {
            div.appendChild(document.createTextNode(sentences[i]));
            div.appendChild(document.createElement("br"));
            div.appendChild(document.createElement("br"));
        }
        div.appendChild(document.createTextNode
                        (serializer.serializeToString(result[i]))
                       );
        div.appendChild(document.createElement("br"));
        div.appendChild(document.createElement("br"));
    }
}
if ("voice" in expression)
{
    result = expression.voice;
    if (typeof result == "string")
    {
        message("Voice error: " + result);
```

```
        }
        else
        {
            if (previous_voice) previous_voice.stop();
            result.speak();
            previous_voice = result;
        }
    }

    formula_display.appendChild(div);
}

function adjust_size()
{
    var available_height;
    if ("innerHeight" in window)
    {
        available_height = window.innerHeight;
    }
    else
    {
        // Fallback for Internet Explorer
        available_height = document.documentElement.clientHeight;
    }
    available_height -= blackboard.offsetTop;
    blackboard.style.height = available_height + "px";
    // This fails in Opera 9
    var palettes = $("palettes");
    if (!palettes) return;

    var font_size = null;
    if (palettes.offsetWidth > palettes.offsetHeight
        && palettes.offsetHeight > available_height
        || palettes.offsetHeight > palettes.offsetWidth
        && palettes.offsetWidth > blackboard.offsetWidth/2)
    {
        font_size = "3mm";
    }
    var node = palettes.firstChild;
    while (node)
    {
        if (node.nodeType == node.ELEMENT_NODE)
        {
            if (font_size) node.style.fontSize = font_size;
```

```
        }
        node = node.nextSibling;
    }
    if (font_size) $("palette-mini").style.fontSize = font_size;
    window.onresize = function (event)
    { window.setTimeout("adjust_size()", 100); };
}

function set_theme(name)
{
    blackboard.className = name;
    tacto.font_changed();
}

function formula_changed(formula)
{
    if (display_timer) window.clearTimeout(display_timer);
    display_timer = window.setTimeout(function () {
        display_formula();
        display_timer = null;
    }, 100);
}

function init()
{
    tacto = new Tacto_editor(blackboard);
    adjust_size();
    tacto.attach_text_input($("keyboard-input"));
    tacto.onchange = formula_changed;
    tacto.options.lock_rotation   = $("lock_rotation").checked;
    tacto.options.live_preview    = $("live_preview").checked;
    if (navigator.platform == 'iPad' || navigator.platform == 'iPod')
    {
        // In iOS v4.2 HTML5 audio playing from JavaScript
        // does not work at all, because Apple disabled it:
        // http://stackoverflow.com/questions/3009888/
        // autoplay-audio-files-on-an-ipad-with-html5
        $("voice-active").disabled = true;
        // Update: This is linked with the pop-up blocker in Safari.
        // If pop-ups are allowed, automatic audio playing works too,
        // but the quality is still atrocious so we disable it in
        // any case.
    }
```

```
    // Initial formula example:
    var x = (blackboard.clientWidth -
             document.getElementById( "palettes" ).clientWidth
            ) / 2 - 100;
    if (x < 10) x = 10;
    tacto.move_to(x, blackboard.clientHeight / 2);
    tacto.insert( "3(x+1)^2_<yz" );
}
```

TACTO implementation in JavaScript

Figure 3: The main source file of the TACTO formula editor

```
// Copyright 2010 Alberto González Palomo http://matracas.org/
// Author: Alberto González Palomo
//
// English:
//   This Program may be used by anyone in accordance with
//   the terms of the German Free Software License.
//
//   The License may be obtained under
//   <http://www.d-fsl.org>
//
// Deutsch:
//   Dieses Programm kann durch jedermann gemäß den Bestimmungen
//   der Deutschen Freien Software Lizenz genutzt werden.
//
//   Die Lizenz kann unter <http://www.d-fsl.de>
//   abgerufen werden.
//
function Tacto_editor(viewport)
{
    var self = this;
    // public:
    self.move_to = move_to;
    self.move_by = move_by;
    self.insert = insert;
    self.do_command = do_command;
    self.zoom = zoom;
    self.get_expression = get_expression;
    self.options =
        {
            "lock_rotation" : true,
            "live_preview" : false,
            "quick_rendering" : true
        };
    self.show_cursor = caret_blink;
    self.font_changed = font_changed;
    self.attach_text_input = function (input)
    {
```

```
        standard_addEventListener(input);
        input.addEventListener('input',  text_input_change, true);
        input.addEventListener('change', text_input_change, true);
        input.addEventListener('focus', function (event)
                               { window.setTimeout
                                 (function()
                                  { self.show_cursor(true);},
                                  100)
                               },
                               true);
        input.addEventListener('blur', function (event)
                               { formula.focus(); },
                               true);
        input.addEventListener('keydown', function (event)
                               {
                                   if (input.value == "")
                                   {
                                       switch (event.keyCode)
                                       {
                                       case 8:  // Backspace
                                       case 46: // Delete
                                       case 37: // Left
                                       case 38: // Up
                                       case 39: // Right
                                       case 40: // Down
                                           keydown(event);
                                       }
                                   }
                               },
                               true);
    };
    // Cross-browser (actually Internet Explorer)
    // compatibility functions:
    self.standard_addEventListener = standard_addEventListener;
    self.standard_event = standard_event;


    // private:
    ////////////////////////////////////////////////////////////////
    // Compatibility functions to abstract browser differences

    function standard_addEventListener(item)
    {
```

```
    if (!item.addEventListener && item.attachEvent)
    {
        // This is for Internet Explorer:
        item.addEventListener = function(type, code, capture)
        {
            // IE does not support event capture.
            item.attachEvent( "on" + type, code);
        }
    }
}

function standard_event(event)
{
    if (!event.target && event.srcElement)
    {
        // This is for Internet Explorer:
        event.target = event.srcElement;
    }
    if (!event.currentTarget)
    {
        event.currentTarget = event.target;
    }
    // We could put here the position values in layer[XY],
    // but in WebKit they are read-only.
    if (!event.preventDefault)
    {
        // This is for Internet Explorer:
        event.preventDefault = function ()
        { event.returnValue = false; }
    }
}

if (!( "Node" in window))
{
    // This is for Internet Explorer:
    // In version 9 it is no longer necessary.
    window.Node =
        {
            // Just the one we need:
            ELEMENT_NODE: 1
            // Others can be added on demand.
        };
}
```

```
function get_text(node)
{
    // This is for Internet Explorer:
    if ("textContent" in node) return node.textContent;
    else                       return node.innerText;
}
function set_text(node, text)
{
    // This is for Internet Explorer:
    if ("textContent" in node) node.textContent = text;
    else                       node.innerText   = text;
}
function append_text(node, text)
{
    // This is for Internet Explorer:
    if ("textContent" in node) node.textContent += text;
    else                       node.innerText   += text;
}

function get_target(event)
{
    // Bug in Safari:
    // http://www.quirksmode.org/js/events_properties.html
    if (event.target.nodeType == event.target.TEXT_NODE)
    {
        return event.target.parentNode;
    }
    else
    {
        return event.target;
    }
}

// Different browsers report different coordinates for events:
// http://hartshorne.ca/2006/01/18/javascript_events/
// Additionally, single-finger touch events still have their
// coordinates inside the touches array.
// This function normalizes all that so that the event handling
// functions receive the same event position in the same
// coordinate system.
function compute_position(event)
{
    var position, x, y;
    if ("touches" in event && event.touches.length > 0)
```

```
{
    x = event.touches.item(0).pageX - viewport.offsetLeft;
    y = event.touches.item(0).pageY - viewport.offsetTop;
}
else if ("touchend" == event.type)
{
    if (!touch) return null;
    x = touch.x;
    y = touch.y;
}
else if ("pageX" in event)
{
    x = event.pageX - viewport.offsetLeft
        + viewport.scrollLeft;
    y = event.pageY - viewport.offsetTop
        + viewport.scrollTop;
}
else
{
    // Internet Explorer
    x = event.x;
    y = event.y;
}
var matrix = null;
if ("webkitTransform" in formula.style)
{
    matrix = window.getComputedStyle
    (formula, null).webkitTransform;
    // Bug in Google Chrome 8.0.552.215 beta:
    // decimal numbers are localized, so we get commas
    // instead of decimal points, which WebKitCSSMatrix
    // can not parse back.
    matrix = matrix.replace(/([0-9]),([0-9])/g, "$1.$2");
}
else if ("MozTransform" in formula.style)
{
    matrix = window.getComputedStyle
    (formula, null).MozTransform;
}
if (matrix && matrix != "none")
{
    matrix = new CSSMatrix(matrix);
    // matrix(a, b, c, d, e, f) = [ a c e ]
    //                            [ b d f ]
```

```
            var f = (matrix.a * matrix.d - matrix.c * matrix.b);
            if (f != 0)
            {
                x = x - matrix.e;
                y = y - matrix.f;
                position =
                    {
                        x: Math.round(( matrix.d * x - matrix.c * y)
                                      / f),
                        y: Math.round((-matrix.b * x + matrix.a * y)
                                      / f)
                    };
            }
            else
            {
                position = { x: Math.round(x - matrix.e),
                             y: Math.round(y - matrix.f) };
            }
        }
        else
        {
            position = { x:x, y:y };
        }

        return position;
    }

    function compute_font_size(element)
    {
        if ( "getComputedStyle" in window)
        {
            var size = window.getComputedStyle(element, null).fontSize;
            return Number(size.replace( /px/ , "" ));
        }
        else
        {
            // This is for Internet Explorer:
            // We can't use element.currentStyle.fontSize
            // because we get back the size we specified in CSS
            // in mm, not pixels.
            return element.offsetHeight;
        }
    }
```

```
if ("WebKitCSSMatrix" in window)
{
    window.CSSMatrix = WebKitCSSMatrix;
}
else if (! ("CSSMatrix" in window))
{
    // matrix(a, b, c, d, e, f) = [ a c e ]
    //                            [ b d f ]
    window.CSSMatrix = function (css_string)
    {
        if (!css_string || css_string == "none")
        {
            this.a = 1; this.c = 0; this.e = 0;
            this.b = 0; this.d = 1; this.f = 0;
        }
        else
        {
            var tokens = css_string.split( /[^0-9.]+/ );
            this.a = tokens[1]; this.c = tokens[3]; this.e = tokens[5];
            this.b = tokens[2]; this.d = tokens[4]; this.f = tokens[6];
        }
        this.toString = function ()
        {
            return ("matrix("
                    + this.a + ", " + this.b + ", "
                    + this.c + ", " + this.d + ", "
                    + this.e + "px, " + this.f + "px)");
        };
    };
}

function disable_selection(node)
{
    // Disable selection:
    node.style.userSelect       = "none";// CSS 3
    node.style.MozUserSelect    = "none";// Gecko
    node.style.webkitUserSelect = "none";// Webkit
    node.style.OUserSelect      = "none";// Opera
    // Internet Explorer:
    node.setAttribute("onselectstart", "return false");
}

// Similar functions are available in DOM Level 3
```

```
function elementChildren(node)
{
    var children = [];
    var n = node.firstChild;
    while (n)
    {
        if (n.nodeType == Node.ELEMENT_NODE) children.push(n);
        n = n.nextSibling;
    }

    return children;
}


////////////////////////////////////////////////////////////////
// Core user interface functionality

var formula = document.createElement( "div" );
viewport.appendChild(formula);
viewport.style.overflow = "hidden";
var handle = { "anchor":null, "sizer":null };
var item_padding = 0;
var item_border  = 0;
var font_pixel_factor = 1;
var font_pixel_factor_h = 1;
var font_pixel_factor_accuracy = 0;
var font_pixel_factor_accuracy_h = 0;
var min_size = 6;
var item_class_name = "tacto-item";
function show_handles(show)
{
    if (show)
    {
        handle.anchor.style.zIndex = "1";
        handle.sizer .style.zIndex = "1";
        handle.anchor.style.visibility = "visible";
        handle.sizer .style.visibility = "visible";
    }
    else
    {
        handle.anchor.style.zIndex = "-1";
        handle.sizer .style.zIndex = "-1";
        handle.anchor.style.visibility = "hidden";
        handle.sizer .style.visibility = "hidden";
```

```
    }
}

function font_changed()
{
    font_pixel_factor_accuracy   = 0;
    font_pixel_factor_accuracy_h = 0;
    select_item(null, false);
}
function compute_font_pixel_factor(font_size, pixel_size)
{
    if (pixel_size > font_pixel_factor_accuracy)
    {
        font_pixel_factor = font_size / pixel_size;
        font_pixel_factor_accuracy = pixel_size;
    }
}
function compute_font_pixel_factor_h(font_size, pixel_size)
{
    if (pixel_size > font_pixel_factor_accuracy_h)
    {
        font_pixel_factor_h = font_size / pixel_size;
        font_pixel_factor_accuracy_h = pixel_size;
    }
}

function move_node(node, x, y)
{
    node.style.left = String(Math.round(x)) + "px";
    node.style.top  = String(Math.round(y)) + "px";
}
function move_nodes_by(nodes, delta_x, delta_y)
{
    for (var i = 0; i < nodes.length; ++i)
    {
        var node = nodes[i];
        var rect = get_rect(node);
        move_node(node, rect.x + delta_x, rect.y + delta_y);
    }
}

function create_handle()
{
```

```
        var handle = document.createElement( "div" );
        handle.setAttribute( "class" , "handle" );
        handle.style.position = "absolute" ;
        handle.style.minWidth  = "1em" ;
        handle.style.minHeight = "1em" ;
        handle.style.border = "medium solid yellow" ;
        handle.style.border.boxSizing       = "border-box" ;
        handle.style.border.MozBoxSizing    = "border-box" ;
        handle.style.border.WebkitBoxSizing = "border-box" ;
        try { handle.style.background = "rgba(255,255,0,0.5)" ; }
        catch (e)
        { /* Alright. This is an old browser like IExplorer 7. */ };
        handle.style.zIndex = "-1" ;
        handle.style.borderRadius       = "1em" ;
        handle.style.MozBorderRadius    = handle.style.borderRadius;
        handle.style.WebkitBorderRadius = handle.style.borderRadius;
        handle.style.cursor = "move" ;
        handle.style.visbility = "hidden" ;

        return handle;
    }
    function insert_handles()
    {
        handle.anchor = create_handle();
        handle.sizer  = create_handle();
        formula.appendChild(handle.anchor);
        formula.appendChild(handle.sizer);
    }
    var move_handles =
        {
            "horizontal" : function (rect)
            {
                var center_x, center_y;
                center_x = rect.x + rect.width /2;
                center_y = rect.y + rect.height/2;

                move_node(handle.anchor,
                          rect.x   - handle.anchor.offsetWidth,
                          center_y - handle.anchor.offsetHeight /2);
                move_node(handle.sizer,
                          rect.x   + rect.width,
                          center_y - handle.sizer.offsetHeight /2);
            },
```

```
        "vertical" : function (rect)
        {
            var center_x, center_y;
            center_x = rect.x + rect.width /2;
            center_y = rect.y + rect.height/2;
            move_node(handle.anchor,
                      center_x - handle.anchor.offsetWidth /2,
                      rect.y   - handle.anchor.offsetHeight);
            move_node(handle.sizer,
                      center_x - handle.sizer.offsetWidth /2,
                      rect.y   + rect.height);
        },
        "both" : function (rect)
        {
            var center_x, center_y;
            center_x = rect.x + rect.width /2;
            center_y = rect.y + rect.height/2;
            move_node(handle.anchor,
                      rect.x - handle.anchor.offsetWidth,
                      rect.y - handle.anchor.offsetHeight);
            move_node(handle.sizer,
                      rect.x + rect.width,
                      rect.y + rect.height);
        }
    };
var stretch_dimension = "vertical";
function set_handles(node)
{
    if (is_item(node.parentNode)) node = node.parentNode;
    if (is_item(node))
    {
        switch (node.getAttribute("stretch"))
        {
        case "horizontal":
            stretch_dimension = "horizontal";
            break;
        case "vertical":
            stretch_dimension = "vertical";
            break;
        default:
            stretch_dimension = "both";
        }
        move_handles[stretch_dimension](get_rect(node));
    }
```

```
}
function get_rect(node)
{
    return {
        "x"      : node.offsetLeft,  "y"      : node.offsetTop,
        "width"  : node.offsetWidth, "height" : node.offsetHeight
    };
}
var get_handle_rect =
    {
        "horizontal" : function ()
        {
            var a = handle.anchor, s = handle.sizer;
            var rect = {
                "x" : a.offsetLeft + a.offsetWidth,
                "y" : a.offsetTop  + a.offsetHeight /2
            };
            rect.width  = (s.offsetLeft
                           - rect.x);
            rect.height = (s.offsetTop
                           + handle.sizer.offsetHeight /2
                           - rect.y);
            return rect;
        },
        "vertical" : function ()
        {
            var a = handle.anchor, s = handle.sizer;
            var rect = {
                "x" : a.offsetLeft + a.offsetWidth /2,
                "y" : a.offsetTop  + a.offsetHeight
            };
            rect.width  = (s.offsetLeft
                           + s.offsetWidth /2
                           - rect.x);
            rect.height = (s.offsetTop
                           - rect.y);
            return rect;
        },
        "both" : function ()
        {
            var a = handle.anchor, s = handle.sizer;
            var rect = {
                "x" : a.offsetLeft + a.offsetWidth,
```

```
                "y" : a.offsetTop  + a.offsetHeight
            };
            rect.width  = (s.offsetLeft
                           - rect.x);
            rect.height = (s.offsetTop
                           - rect.y);
            return rect;
        }
    };


var scale_items = {
    "horizontal" : function (nodes, rect)
    {
        var node = nodes[0];
        var previous_rect = get_rect(node);
        if (!rect) rect = get_handle_rect.horizontal();
        var new_height = rect.width * font_pixel_factor;
        node.style.fontSize = String(new_height) + "px";
        var stretch_scale = (line_height / rect.width);
        transform_item(node,
                       "scale(1.0, " + stretch_scale + ")");
        if (node.offsetWidth != rect.width)
        {
            compute_font_pixel_factor_h(new_height,
                                        node.offsetWidth);
            new_height = rect.width * font_pixel_factor_h;
            node.style.fontSize = String(new_height) + "px";
        }
        node.setAttribute("scaled-height",
                          node.offsetHeight * stretch_scale);
        rect.y -= node.clientHeight/2;
        move_nodes_by(nodes,
                      rect.x - previous_rect.x,
                      rect.y - previous_rect.y);
    },
    "vertical" : function (nodes, rect)
    {
        var node = nodes[0];
        var previous_rect = get_rect(node);
        if (!rect) rect = get_handle_rect.vertical();
        var new_height = rect.height * font_pixel_factor;
        node.style.fontSize = String(new_height) + "px";
        var stretch_scale = (line_height / rect.height);
```

```
            transform_item(node,
                           "scale(" + stretch_scale + ", 1.0)");
            if (node.offsetHeight != rect.height)
            {
                compute_font_pixel_factor(new_height,
                                          node.offsetHeight);
                new_height = rect.height * font_pixel_factor;
                node.style.fontSize = String(new_height) + "px";
            }
            node.setAttribute("scaled-width",
                              node.offsetWidth * stretch_scale);
            rect.x -= node.clientWidth /2;
            move_nodes_by(nodes,
                          rect.x - previous_rect.x,
                          rect.y - previous_rect.y);
        },
        "both": function (nodes, rect)
        {
            var node = nodes[0];
            var previous_rect = get_rect(node);
            if (!rect) rect = get_handle_rect.both();
            var new_height = rect.height * font_pixel_factor;
            node.style.fontSize = String(new_height);// + "px";
            if (node.offsetHeight != rect.height)
            {
                compute_font_pixel_factor(new_height,
                                          node.offsetHeight);
                new_height = rect.height * font_pixel_factor;
                node.style.fontSize = String(new_height) + "px";
            }
            move_nodes_by(nodes,
                          rect.x - previous_rect.x,
                          rect.y - previous_rect.y);
        }
    };
    var item_tag_name = "div"
    function is_item(node)
    {
        return (node
                && node.nodeType == Node.ELEMENT_NODE
                && node.className == item_class_name);
    }
    var caret =
        {
```

```
        node: document.createElement( "div" ),
        x: 0,  y: 0,
        center_y: 0,
        state: 0,
        move_to: function (x, y)
        {
            this.x = x;
            this.y = y;
            move_node(this.node,
                      this.x,
                      this.y - this.center_y);
        }
    };
caret.node.setAttribute( "id" , "tacto-cursor" );
var line_height = 20;// pixels
var opacity_available = "opacity" in document.body.style;
caret.node
    .setAttribute( "style" ,
                   "position:absolute;padding:0;margin:0;"
                   + "width:0px;cursor:text;opacity:0;" );
caret.node.appendChild(document.createTextNode( "|" ));
function caret_blink(reset)
{
    if (tacto_has_focus && true == reset)
    {
        caret.state = 10;
        if (!caret_blink_interval)
        {
            window.caret_blink = caret_blink;
            caret_blink_interval = window
                .setInterval( "caret_blink()" , 100);
        }
    }
    else if (!tacto_has_focus || false == reset)
    {
        if (reset == false) caret.state = 0;
        else                caret.state = 10;
        if (caret_blink_interval)
        {
            window.clearInterval(caret_blink_interval);
            caret_blink_interval = null;
        }
    }
```

```
        if (caret.node.firstChild)
        {
            // Compute the font metrics we need:
            line_height = caret.node.offsetHeight;
            min_size = line_height / 2;
            var font_size = compute_font_size(caret.node);
            compute_font_pixel_factor(font_size, line_height);
            cursor_step           = line_height / 3;
            caret.center_y = line_height / 2;
            caret.node.style.height = String(line_height) + "px";
            caret.node.removeChild(caret.node.firstChild);
        }
        if (opacity_available)
        {
            caret.node.style.opacity = caret.state/10;
        }
        else
        {
            caret.node.style.borderColor =
                (caret.state > 5? "yellow" : "transparent");
        }
        if (caret.state > 0) --caret.state;
        else caret.state = 10;
    }
    window.caret_blink = caret_blink;
    var caret_blink_interval;
    formula.appendChild(caret.node);
    var cursor_step = 0;

    function move_to(x, y)
    {
        if (!isNaN(x) && !isNaN(y))
        {
            if (x < 0) x = 0;// TODO: expand also to the left/top.
            if (y < 0) y = 0;
            else
            {
                y = Math.round(y / cursor_step) * cursor_step;
            }
            if (formula.offsetWidth  < x+cursor_step)
            {
                formula.style.width  = String(x + 4*cursor_step)+"px";
            }
            if (formula.offsetHeight < y+cursor_step)
```

```
            {
                formula.style.height = String(y + 4*cursor_step)+"px";
            }
            caret.move_to(x, y);
            //caret.scrollIntoView();// This blinks in Firefox
        }
    }
    function move_by(delta_x, delta_y)
    {
        move_to(caret.x + delta_x, caret.y + delta_y);
    }
    function insert(content)
    {
        var item;
        if (content.length > 1)
        {
            for (var i = 0; i < content.length; ++i)
            {
                item = insert(content[i]);
            }
            return item;
        }
        switch(content)
        {
        case "^": move_by(0, -cursor_step); return null;
        case "_": move_by(0,  cursor_step); return null;
        case " ": move_by(cursor_step,  0); return null;
        case "":                            return null;
        }
        var x, y;
        x = caret.x;
        y = caret.y - caret.center_y;
        item = document.createElement(item_tag_name);
        item.className = item_class_name;
        item.style.position = "absolute";
        move_node(item, x, y);
        item.style.padding = "0";
        item.style.overflow = "hidden";
        item.style.cursor = "move";
        disable_selection(item);
        if (content == "/") content = "÷";
        if (content == "÷")
        {
```

```
            content = "\u2014";// em dash
            item.setAttribute("stretch", "horizontal");
        }
        else if (content.match(/[(){}\[\]∫]/))
        {
            item.setAttribute("stretch", "vertical");
        }
        if (item_tag_name != "input")
        {
            item.appendChild(document.createTextNode(content));
        }
        else
        {
            item.setAttribute("value", content);
            item.style.background = "transparent";
            item.style.border = "none";
            item.style.overflow = "visible";
            item.style.width = "auto";
            item.focus();
        }
        formula.appendChild(item);
        select_item(item, false);
        move_by(item.offsetWidth, 0);
        update();
        if (self.onchange) self.onchange(self);

        return item;
    }
    function get_item_at(position)
    {
        var selected_symbol = tree.symbol_at(position);
        if (selected_symbol) return selected_symbol.node;
        else                 return null;
    }
    function select_item_at(position, add_to_selection)
    {
        var node = get_item_at(position);
        if (node && node.hasAttribute("selected")) return;
        select_item(node, add_to_selection);
    }
    function select_items_inside(path, add_to_selection)
    {
        var symbols_inside_path = tree.symbols_inside(path);
```

```
    for (var i = 0; i < symbols_inside_path.length; ++i)
    {
        select_item(symbols_inside_path[i].node, add_to_selection);
        add_to_selection = true;
    }
}
function select_item(item, add_to_selection)
{
    if (item && !is_item(item) && is_item(item.parentNode))
    {
        item = item.parentNode;
    }
    var already_selected = false;
    var i;
    i = 0;
    while (i < selected_nodes.length)
    {
        if (selected_nodes[i] == item)
        {
            already_selected = true;
            selected_nodes.splice(i, 1);
        }
        else ++i;
    }

    var node = formula.firstChild;
    while (node)
    {
        if (is_item(node)) node.removeAttribute("selected");
        node = node.nextSibling;
    }
    if (!add_to_selection) selected_nodes = [];
    if (item && !already_selected) selected_nodes.push(item);

    if (selected_nodes.length)
    {
        for (i = 0; i < selected_nodes.length; ++i)
        {
            selected_nodes[i].setAttribute("selected",
                                           "selected");
        }
        set_handles(selected_nodes[0]);
        // Disable the manual scaling of numbers etc. for now,
        // as it just confuses users.
```

```
            // The handles will be shown only for fraction bars,
            // parentheses and similar things that need to be
            // stretched.
            show_handles(1 == selected_nodes.length
                         && stretch_dimension != "both");
        }
        else
        {
            show_handles(false);
        }
    }
    function del(count)
    {
        if (undefined == count) count = 1;
        var to_delete, abs_count;
        if (selected_nodes.length > 0)
        {
            to_delete = [];
            for (var i = 0; i < selected_nodes.length; ++i)
            {
                to_delete.push(new BSTNode(selected_nodes[i]));
            }
            abs_count = to_delete.length;
            select_item(null, false);
        }
        else
        {
            to_delete = [];
            var rect, x_min, y_min, x_max, y_max;
            x_min = caret.x;
            y_min = caret.y - cursor_step;
            move_by(count * cursor_step, 0);
            x_max = caret.x;
            y_max = caret.y + cursor_step + line_height;
            if (count < 0)
            {
                var temp = x_min;
                x_min = x_max;
                x_max = temp;
            }

            var symbols = get_symbols();
            var i, symbol;
            for (i = 0; i < symbols.length; ++i)
```

```
            {
                symbol = symbols[i];
                if (symbol.centroid_x <= x_max &&
                    symbol.centroid_x >= x_min &&
                    symbol.centroid_y <= y_max &&
                    symbol.centroid_y >= y_min)
                {
                    to_delete.push(symbol);
                }
            }
            if (to_delete.length < 1) return;

            if (count > 0)
            {
                abs_count = count;
                to_delete.sort(function (a, b)
                               { return a.min_x - b.min_x; }
                              );
            }
            else
            {
                abs_count = -count;
                to_delete.sort(function (a, b)
                               { return b.max_x - a.max_x; }
                              );
            }
        }

        abs_count = Math.min(abs_count, to_delete.length);
        for (i = 0; i < abs_count; ++i)
        {
            symbol = to_delete[i];
            if (count > 0) move_to(symbol.max_x, symbol.centroid_y);
            else           move_to(symbol.min_x, symbol.centroid_y);
            formula.removeChild(symbol.node);
        }
        update();
        if (self.onchange) self.onchange(self);
    }

    var dragging_node = null;
    var drag_offset = { x:0, y:0 };
    function drag_start(node, position)
    {
```

```
        caret_blink(false);
        if (is_item(node))
        {
            var rect = get_rect(node);
            drag_offset.x = position.x - (rect.x + rect.width /2);
            drag_offset.y = position.y - (rect.y + rect.height/2);
            dragging_node = node;
        }
        else
        {
            drag_start_handle[stretch_dimension](node, position);
        }
    }
    var drag_start_handle =
        {
            horizontal: function (node, position)
            {
                var rect = get_rect(node);
                if (node == handle.anchor)
                {
                    drag_offset.x =
                        position.x - (rect.x + rect.width    );
                    drag_offset.y =
                        position.y - (rect.y + rect.height /2);
                    dragging_node = node;
                }
                else if (node == handle.sizer)
                {
                    drag_offset.x =
                        position.x - (rect.x                 );
                    drag_offset.y =
                        position.y - (rect.y + rect.height /2);
                    dragging_node = node;
                }
            },
            vertical: function (node, position)
            {
                var rect = get_rect(node);
                if (node == handle.anchor)
                {
                    drag_offset.x =
                        position.x - (rect.x + rect.width /2);
                    drag_offset.y =
                        position.y - (rect.y + rect.height  );
```

```
                dragging_node = node;
            }
            else if (node == handle.sizer)
            {
                drag_offset.x =
                    position.x - (rect.x + rect.width  /2);
                drag_offset.y =
                    position.y - (rect.y                );
                dragging_node = node;
            }
        },
        both: function (node, position)
        {
            var rect = get_rect(node);
            if (node == handle.anchor)
            {
                drag_offset.x =
                    position.x - (rect.x + rect.width    );
                drag_offset.y =
                    position.y - (rect.y + rect.height   );
                dragging_node = node;
            }
            else if (node == handle.sizer)
            {
                drag_offset.x =
                    position.x - (rect.x);
                drag_offset.y =
                    position.y - (rect.y);
                dragging_node = node;
            }
        }
    };
var drag_move_handle =
    {
        horizontal: function (rect, position)
        {
            var delta_x = position.x - drag_offset.x;
            var delta_y = position.y - drag_offset.y;
            if (dragging_node == handle.anchor)
            {
                rect.width  = rect.x + rect.width  - delta_x;
                rect.height = rect.y + rect.height - delta_y;
                rect.x = position.x-drag_offset.x;
                rect.y = position.y-drag_offset.y;
```

```
                rect.width = Math.max(rect.width, min_size);
            }
            else if (dragging_node == handle.sizer)
            {
                rect.width  = delta_x - rect.x;
                rect.height = delta_y - rect.y;
                rect.y = position.y-drag_offset.y;
                if (rect.width  < min_size)
                {
                    rect.x -= min_size - rect.width;
                    rect.width  = min_size;
                }
            }
        },
        vertical: function (rect, position)
        {
            var delta_x = position.x - drag_offset.x;
            var delta_y = position.y - drag_offset.y;
            if (dragging_node == handle.anchor)
            {
                rect.width  = rect.x + rect.width  - delta_x;
                rect.height = rect.y + rect.height - delta_y;
                rect.x = position.x-drag_offset.x;
                rect.y = position.y-drag_offset.y;
                rect.height = Math.max(rect.height, min_size);
            }
            else if (dragging_node == handle.sizer)
            {
                rect.width  = delta_x - rect.x;
                rect.height = delta_y - rect.y;
                rect.x = position.x-drag_offset.x;
                if (rect.height < min_size)
                {
                    rect.y -= min_size - rect.height;
                    rect.height = min_size;
                }
            }
        },
        both: function (rect, position)
        {
            var delta_x = position.x - drag_offset.x;
            var delta_y = position.y - drag_offset.y;
            if (dragging_node == handle.anchor)
            {
```

```
                rect.width  = rect.x + rect.width  - delta_x;
                rect.height = rect.y + rect.height - delta_y;
                rect.x = delta_x;
                rect.y = delta_y;
                rect.width  = Math.max(rect.width,  min_size);
                rect.height = Math.max(rect.height, min_size);
            }
            else if (dragging_node == handle.sizer)
            {
                rect.width  = delta_x - rect.x;
                rect.height = delta_y - rect.y;
                if (rect.width  < min_size)
                {
                    rect.x -= min_size - rect.width;
                    rect.width  = min_size;
                }
                if (rect.height < min_size)
                {
                    rect.y -= min_size - rect.height;
                    rect.height = min_size;
                }
            }
        }
    };
function drag_move(position)
{
    if (dragging_node)
    {
        var rect = get_handle_rect[stretch_dimension]();
        var height = rect.height;
        if (is_item(dragging_node))
        {
            rect.x = (position.x - rect.width /2
                      - drag_offset.x);
            rect.y = (position.y - rect.height/2
                      - drag_offset.y);
        }
        else
        {
            drag_move_handle[stretch_dimension](rect,
                                                position);
        }
        move_handles[stretch_dimension](rect);
        if (selected_nodes.length > 0)
```

```
            {
                scale_items[stretch_dimension](selected_nodes,
                                               rect);
            }
            if (self.options.live_preview)
            {
                update();
                if (self.onchange) self.onchange(self);
            }
        }
    }
    function drag_cancel()
    {
        dragging_node = null;
        caret_blink(true);
    }
    function drag_end()
    {
        if (dragging_node && selected_nodes.length > 0)
        {
            scale_items[stretch_dimension](selected_nodes, null);
            set_handles(selected_nodes[0]);
            update();
            if (self.onchange) self.onchange(self);
        }
        drag_cancel();
    }

    var gesture_path =
        {
            node:   undefined,
            points: undefined,
            // Bounding box:
            x:      Number.MAX_VALUE, y:       Number.MAX_VALUE,
            width: -Number.MAX_VALUE, height: -Number.MAX_VALUE,
            init: function(point)
            {
                this.x = point.x; this.width  = 0;
                this.y = point.y; this.height = 0;
                if ("pathSegList" in this.node)
                {
                    this.points = [];
                    this.node.pathSegList.clear();
                    this.extend(point);
```

```
    }
    else
    {
        this.update_html_rectangle();
        this.node.style.display = "block";
    }
    this.node.style.visibility = "visible";
},
update_html_rectangle: function ()
{
    var style = this.node.style;
    style.left   = String(this.x) + "px";
    style.top    = String(this.y) + "px";
    style.width  = String(this.width)  + "px";
    style.height = String(this.height) + "px";
},
extend: function (point)
{
    if (point.x < this.x)
    {
        this.width += this.x - point.x;
        this.x = point.x;
    }
    else if (point.x > this.x + this.width)
    {
        this.width += (point.x
                       - (this.x + this.width));
    }
    if (point.y < this.y)
    {
        this.height += this.y - point.y;
        this.y = point.y;
    }
    else if (point.y > this.y + this.height)
    {
        this.height += (point.y
                        - (this.y + this.height));
    }
    if (this.points)
    {
        this.points.push(point);
        var segment;
        if (this.points.length > 1)
```

```
                {
                    segment = this.node.
                        createSVGPathSegLinetoAbs(point.x,
                                                  point.y);
                }
                else
                {
                    segment = this.node.
                        createSVGPathSegMovetoAbs(point.x,
                                                  point.y);
                }
                this.node.pathSegList.appendItem(segment);
            }
            else
            {
                this.update_html_rectangle();
            }
        },
        contains: function (x, y)
        {
            var inside_bounding_box =
                (x >= this.x && x <= this.x + this.width &&
                 y >= this.y && y <= this.y + this.height);
            if (inside_bounding_box && this.points)
            {
                // Compute intersection count
                var intersection_count = 0, a, b;
                a = this.points[this.points.length-1]
                for (var i = 0; i < this.points.length; ++i)
                {
                    b = this.points[i];
                    var ac_y = y - a.y, cb_y = b.y - y;
                    var ac_x = x - a.x, cb_x = b.x - x;
                    if (cb_y == 0)
                    {
                        if (x >= b.x) ++intersection_count;
                    }
                    else if (ac_y > 0 && cb_y > 0
                             && ac_x/ac_y > cb_x/cb_y)
                    {
                        ++intersection_count;
                    }
                    else if (ac_y < 0 && cb_y < 0
                             && ac_x/ac_y < cb_x/cb_y)
```

```
                    {
                        ++intersection_count;
                    }

                    a = b;
                }

                return (intersection_count & 0x01 != 0x00);
            }
            else
            {
                return inside_bounding_box;
            }
        },
        clear: function ()
        {
            if (this.points) this.node.pathSegList.clear();
            this.node.style.visibility = "hidden";
        }
    };
try
{
    gesture_path.node = document.
        createElementNS("http://www.w3.org/2000/svg",
                        "path");
}
catch (e)
{
    // The browser does not support embedded SVG.
    gesture_path.node = document.createElement("div");
};
if ("pathSegList" in gesture_path.node)
{
    gesture_path.node.setAttribute
    ("style", "fill:none;stroke:yellow;stroke-width:2");
    var svg = document.createElementNS
    ("http://www.w3.org/2000/svg", "svg");
    svg.setAttribute("style", "width:100%; height:100%");
    svg.appendChild(gesture_path.node);
    formula.appendChild(svg);
}
else
{
```

```
        gesture_path.node.
            setAttribute("style",
                         "position:absolute;"
                         + "display:none;padding:0;margin:0;"
                         + "border:thin solid yellow;");
        formula.appendChild(gesture_path.node);
    }

    function path_start(position)
    {
        caret_blink(false);
        gesture_path.init(position);
    }
    function path_add(position)
    {
        gesture_path.extend(position);
    }
    function path_cancel()
    {
        gesture_path.clear();
        caret_blink(true);
    }


    function path_end()
    {
        if (gesture_path.points && gesture_path.points.length)
        {
            // Recognize single-touch gestures
            gesture(gesture_path);
        }
        else
        {
            // This is rectangle selection
            select_items_inside(gesture_path, false);
        }
        path_cancel();
    }

    var previous_transformation = "";
    function zoom(scale)
    {
        message("zoom " + scale);
        transform(0, 0, 0, 0, scale, 0);
```

```
        fix();
    }
    this.zoom = zoom;
    function transform_item(node, transform)
    {
        node.style.transform = transform;
        node.style.webkitTransform = node.style.transform;
        node.style.MozTransform    = node.style.transform;
    }
    function transform(center_x, center_y, x, y, scale, rotation)
    {
        if (self.options.lock_rotation) rotation = 0;
        if ("webkitTransformOrigin" in formula.style)
        {
            var translate_open, translate_close;
            if (self.options.quick_rendering)
            {
                translate_open  = "translate3d(";
                translate_close = ", 0)";
            }
            else
            {
                translate_open  = "translate(";
                translate_close = ")";
            }
            formula.style.webkitTransformOrigin = "0 0";
            formula.style.webkitTransform =
                translate_open
                + (center_x + x) + "px, "
                + (center_y + y) + "px" + translate_close
                + (scale != 1.0? " scale(" + (scale) + ")" : "")
                + (rotation? " rotate(" + (rotation) + "deg)" : "")
                + " " + translate_open
                + (-center_x) + "px, "
                + (-center_y) + "px" + translate_close
                + " " + previous_transformation;
        }
        else if ("MozTransformOrigin" in formula.style)
        {
            formula.style.MozTransformOrigin = "0 0";
            formula.style.MozTransform =
                "translate("
```

```
                + (center_x + x) + "px, "
                + (center_y + y) + "px)"
                + (scale != 1.0? " scale(" + (scale) + ")" : "" )
                + (rotation? " rotate(" + (rotation) + "deg)" : "" )
                + " translate("
                + (-center_x) + "px, "
                + (-center_y) + "px)"
                + " " + previous_transformation;
        }
        else
        {
            formula.style.zoom = scale;
        }
    }
    function fix()
    {
        if ("webkitTransform" in formula.style)
        {
            previous_transformation = window
                .getComputedStyle(formula, null)
                .webkitTransform;
        }
        else if ("MozTransform" in formula.style)
        {
            previous_transformation = window
                .getComputedStyle(formula, null)
                .MozTransform;
        }
    }

    ////////////////////////////////////////////////////////////////
    // Commands

    function do_command(command)
    {
        switch (command)
        {
        case "up":        move_by(             0, -cursor_step); break;
        case "down":      move_by(             0,  cursor_step); break;
        case "left":      move_by( -cursor_step,             0); break;
        case "right":     move_by(  cursor_step,             0); break;
        case "backtab":   move_by(8*-cursor_step,            0); break;
```

```
    case "tab":       move_by(8* cursor_step,           0); break;
    case "delete":    del( 1);                              break;
    case "backspace": del(-1);                              break;
    case "baseline":  move_by(            0,  cursor_step); break;
    }
}

////////////////////////////////////////////////////////////////
// Gesture recognition



function gesture(path)
{
    // Recognize gestures
    var first_point = path.points[0];
    var last_point  = path.points[path.points.length - 1];
    var tolerance = 5;
    if (path.width > line_height/2 &&
        first_point.x <=  path.x + tolerance &&
        last_point.x  >= (path.x + path.width - tolerance) &&
        path.width / path.height > 5)
    {
        // Fraction bar gesture
        var item = insert("÷");
        scale_items[item.getAttribute("stretch")]([item], path);
        set_handles(item);
    }
    else if (path.width < 2 && path.height < 2)
    {
        // Click selection
        select_item_at(path, true)
        move_to(path.x, path.y);
    }
    else
    {
        // Lasso selection gesture
        select_items_inside(gesture_path, true);
    }
}

////////////////////////////////////////////////////////////////
// Natural language generation
```

```
var english_infix =
    {
        "arith1:plus" : " plus ",
        "arith1:minus" : " minus ",
        "arith1:times" : " times ",
        "arith1:divide" : " divided by ",
        "arith1:power" : " to the power of ",
        "relation1:eq" : " equals ",
        "relation1:gt" : " is greater than ",
        "relation1:lt" : " is less than ",
        "nums1:rational" : " divided by ",
        "nums1:factorial" : " factorial ",
        "algebra1:vector_selector" : " sub-",
        "matracas.org:mixed-number" : " and "
    };
var english_function =
    {
        "transc1:sin" : " sine of ",
        "transc1:cos" : " cosine of "
    };
var english_symbols =
    {
        "nums1:pi" : " pi "
    };

var number_words = {
    "0":"zero", "1":"one", "2":"two",
    "3":"three", "4":"four", "5":"five",
    "6":"six", "7":"seven", "8":"eight",
    "9":"nine", "10":"ten", "11":"eleven",
    "12":"twelve", "13":"thirteen",
    "14":"fourteen", "15":"fifteen",
    "16":"sixteen", "17":"seventeen",
    "18":"eighteen", "19":"nineteen",
    "20":"twenty", "30":"thirty",
    "40":"forty", "50":"fifty",
    "60":"sixty", "70":"seventy",
    "80":"eighty", "90":"ninety"
};
function om_to_english(om)
```

```
{
    var text = "";

    var i, children;
    switch (om.localName)
    {
    case "OMOBJ":
        text += om_to_english(om.firstChild);
        break;
    case "OMI":
        text += number_to_english(om.firstChild.nodeValue);
        break;
    case "OMF":
        text += number_to_english(om.getAttribute("dec"));
        break;
    case "OMV":
    case "OMS":
        text += om.getAttribute("name");
        break;
    case "OME":
        text += "error";
        break;
    case "OMA":
        children = elementChildren(om);
        if (children.length < 1) break;
        if (children[0].localName == "OMV")
        {
            text += children[0].getAttribute("name")
                + " of";
        }
        else
        {
            var symbol_name
                = children[0].getAttribute("cd") + ":"
                + children[0].getAttribute("name");
            if (symbol_name == "arith1:power")
            {
                var exponent = om_to_english(children[2]);
                if (exponent == "two")
                {
                    text += (om_to_english(children[1])
                             + " squared");
                }
                else if (exponent == "three")
```

```
        {
            text += (om_to_english(children[1])
                     + " cubed");
        }
        else
        {
            text += (om_to_english(children[1])
                     + english_infix[symbol_name]
                     + exponent
                    );
        }
    }
    else if (symbol_name == "algebra1:vector_selector")
    {
        var index = om_to_english(children[2]);
        text += (om_to_english(children[1])
                 + english_infix[symbol_name]
                 + index
                );
    }
    else if (symbol_name == "arith1:root")
    {
        var degree = om_to_english(children[2]);
        if (degree in english_ordinal)
        {
            degree = english_ordinal[degree];
        }
        else
        {
            degree += "-th";
        }
        if (degree == "two")
        {
            text += ("square root of "
                     + om_to_english(children[1])
                    );
        }
        else if (degree == "three")
        {
            text += ("cubic root of "
                     + om_to_english(children[1])
                    );
        }
        else
```

```
                {
                    text += (degree
                             + english_infix[symbol_name]
                             + om_to_english(children[2])
                            );
                }
            }
            else if (symbol_name == "arith1:times"
                     && children[2].nodeName == "OMV")
            {
                text += (om_to_english(children[1])
                         + " "
                         + om_to_english(children[2])
                        );
            }
            else if (symbol_name in english_infix)
            {
                if (children.length > 2)
                {
                    text += (om_to_english(children[1])
                             + english_infix[symbol_name]
                             + om_to_english(children[2])
                            );
                }
                else
                {
                    text += (om_to_english(children[1])
                             + english_infix[symbol_name]
                            );
                }
            }
            else if (symbol_name in english_function)
            {
                text += english_function[symbol_name];
                for (i = 1; i < children.length; ++i)
                {
                    if (i > 1) text += ", ";
                    text += om_to_english(children[i]);
                }
            }
        }
        break;
    }
```

```
    return text;
}

function number_to_english(digits)
{
    var text;
    var parts = digits.split( "." );
    text = integer_to_english(parts[0]);
    if (parts.length > 1)
    {
        text += " point ";
        for (var i = 1; i < parts.length; ++i)
        {
            text += integer_to_english(parts[i]);
        }
    }

    return text;
}

function integer_to_english(digits)
{
    var text = "";
    if (digits.length > 6)
    {
        text +=
        integer_to_english(digits.
                           substring(0, digits.length - 6))
            + " million "
            + integer_to_english(digits.
                                 substring(digits.length - 6));
    }
    else if (digits.length > 3)
    {
        text +=
        integer_to_english(digits.
                           substring(0, digits.length - 3))
            + " thousand "
            + integer_to_english(digits.
                                 substring(digits.length - 3));
    }
    else if (digits.length > 2)
    {
        text +=
```

```
            integer_to_english(digits.
                                substring(0, digits.length - 2))
                + " hundred "
                + integer_to_english(digits.
                                     substring(digits.length - 2));
        }
        else if (digits.length > 1)
        {
            if (digits in number_words)
            {
                text += number_words[digits];
            }
            else
            {
                text += number_words[digits[0]+"0"];
                if (digits[1] != 0)
                {
                    text += "-" + integer_to_english(digits[1]);
                }
            }
        }
        else
        {
            for (var i = 0; i < digits.length; ++i)
            {
                if (i > 0) text += " ";
                text += number_words[digits[i]];
            }
        }

        return text;
    }

    ////////////////////////////////////////////////////////////////
    // Speech synthesis



    function Voice()
    {
        var self = this;

        // Public:
        this.prepare = prepare;
```

```
this.speak = speak;
this.stop  = stop;

// Private:
var sequence;
var current = 0;


var voice_metadata =
    {
        // variant: { token: duration in milliseconds }
        "medial" :
        {
            "and-short" : 140, "and" : 141,
            "a" : 392, "b" : 488,
            "cosine-of" : 693, "cubed" : 442,
            "c" : 559, "divided-by" : 736,
            "d" : 489, "eighteen" : 582,
            "eight" : 236, "eighty" : 325,
            "eleven" : 427, "equals" : 473,
            "e" : 401, "factorial" : 680,
            "fifteen" : 561, "fifty" : 450,
            "five" : 399, "forty" : 447,
            "fourteen" : 587, "four" : 350,
            "f" : 502, "g" : 573,
            "hundred" : 439, "h" : 537,
            "is-greater-than" : 886, "is-less-than" : 760,
            "i" : 463, "j" : 599,
            "k" : 554, "l" : 471,
            "million" : 478, "minus" : 461,
            "m" : 477, "nineteen" : 592,
            "ninety" : 498, "nine" : 436,
            "n" : 465, "one" : 396,
            "o" : 421, "pi" : 372,
            "plus" : 427, "point" : 460,
            "p" : 562, "q" : 581,
            "r" : 473, "seventeen" : 644,
            "seventy" : 507, "seven" : 477,
            "sine-of" : 553, "sixteen" : 585,
            "sixty" : 466, "six" : 434,
            "squared" : 468, "square-root-of" : 841,
```

```
        "sub" : 333, "s" : 458,
        "ten" : 303, "thirteen" : 542,
        "thirty" : 416, "thousand" : 567,
        "three" : 404, "times" : 442,
        "to-the-power-of" : 826, "t" : 482,
        "twelve" : 454, "twenty" : 461,
        "two" : 271, "u" : 523,
        "v" : 521, "w" : 742,
        "x" : 517, "y" : 300,
        "zero" : 490, "z" : 534
    },

    "final" :
    {
        "and" : 312, "a" : 372,
        "b" : 461, "cosine-of" : 723,
        "cubed" : 418, "c" : 515,
        "divided-by" : 669, "d" : 454,
        "eighteen" : 371, "eight" : 310,
        "eighty" : 422, "eleven" : 520,
        "equals" : 1232, "e" : 356,
        "factorial" : 781, "fifteen" : 627,
        "fifty" : 549, "five" : 596,
        "forty" : 531, "fourteen" : 616,
        "four" : 407, "f" : 469,
        "g" : 488, "hundred" : 504,
        "h" : 507, "is-greater-than" : 1519,
        "is-less-than" : 1386, "i" : 421,
        "j" : 534, "k" : 487,
        "l" : 436, "million" : 496,
        "minus" : 412, "m" : 437,
        "nineteen" : 612, "ninety" : 586,
        "nine" : 423, "n" : 407,
        "one" : 323, "o" : 337,
        "pi" : 414, "plus" : 299,
        "point" : 380, "p" : 350,
        "q" : 509, "r" : 411,
        "seventeen" : 723, "seventy" : 669,
        "seven" : 500, "sine-of" : 537,
        "sixteen" : 627, "sixty" : 593,
```

```
                "six": 415, "squared": 491,
                "square-root-of": 747, "sub": 285,
                "s": 405, "ten": 384,
                "thirteen": 599, "thirty": 506,
                "thousand": 582, "three": 407,
                "times": 409, "to-the-power-of": 733,
                "t": 435, "twelve": 486,
                "twenty": 569, "two": 305,
                "u": 454, "v": 460,
                "w": 685, "x": 439,
                "y": 252, "zero": 516,
                "z": 473
            }

        };

    var max_token_length = 0;
    var i;
    for (i in voice_metadata["medial"])
    {
        if (i.length > max_token_length)
        {
            max_token_length = i.length;
        }
    }

    function prepare(text)
    {
        text = text.replace(/[ ]+/g, "-");
        sequence = [];
        var tokens = [];
        var known_tokens = voice_metadata["medial"];
        var start = 0, length;
        for (start = 0; start < text.length; ++start)
        {
            if (text.charCodeAt(start) == 0x2D) continue;
            for (var length = max_token_length;
                 length > 0; --length)
            {
                var token = text.substr(start, length);
                if (token in known_tokens)
                {
```

```
                tokens.push(token);
                start += length - 1;
                break;
            }
        }
        if (length == 0) tokens.push(text.substr(start, 1));
    }
    for (var i = 0; i < tokens.length; ++i)
    {
        if (i+1 < tokens.length) add(tokens[i], "medial");
        else                     add(tokens[i], "final");
    }
}

function speak()
{
    current = 0;
    // Initial delay, needed by Safari:
    var initial_delay = 10;
    if (navigator.platform == 'iPad'
        || navigator.platform == 'iPod')
    {
        // Ugly hack, but it does not work otherwise.
        // In Desktop Safari it works fine.
        initial_delay = 1200;
    }

    voice_sample_timer = setTimeout(next, initial_delay);
}

function stop()
{
    if (voice_sample_timer) clearTimeout(voice_sample_timer);
}

var voice_sample_timer;
function next()
{
    if (current < sequence.length)
    {
        sequence[current][0].play();
        duration = sequence[current][1];
        if (duration > 10 && voice_sample_timer)
        {
```

```
                voice_sample_timer = setTimeout(next, duration);
            }
            ++current;
        }
    }

    function add(token, variant)
    {
        if (!(token in voice_metadata[variant]))
        {
            message("Error: speech token '" + token
                    + "' unavailable");
            return;
        }
        var sample = new Audio("voices/"
                               + variant + "/"
                               + token + ".wav");
        sample.load();
        sample.addEventListener
        ("ended",
         function ()
         {
             this.currentTime = 0;
             this.pause();
         },
         false);
        sequence.push([sample, voice_metadata[variant][token]]);
    }
}


////////////////////////////////////////////////////////////////
// Event handling

var selected_nodes = [];
var mousedown_position;
var modes = { NONE:0, DRAG:1, GESTURE:2 };
var mode = modes.NONE;
function mousedown(event)
{
    standard_event(event);
    var node = get_target(event);
    if (node.className == "handle"
```

```
        && node.style.visibility != "hidden")
    {
        mousedown_position = compute_position(event);
        mode = modes.DRAG;
        drag_start(node, mousedown_position);
    }
    else
    {
        if (node == formula || node.parentNode == formula)
        {
            mousedown_position = compute_position(event);
            select_item_at(mousedown_position, event.shiftKey);
            if (selected_nodes.length > 0)
            {
                mode = modes.DRAG;
                drag_start(selected_nodes[0],
                           mousedown_position);
            }
            else
            {
                mode = modes.GESTURE;
                path_start(mousedown_position);
            }
        }
    }
}
function mousemove(event)
{
    switch (mode)
    {
    case modes.DRAG:
        standard_event(event);
        var position = compute_position(event);
        drag_move(position);
        break;
    case modes.GESTURE:
        standard_event(event);
        path_add(compute_position(event));
        break;
    }
}
function mouseup(event)
{
    if (mousedown_position)
```

```
        {
            standard_event(event);
            var position = compute_position(event);
            if (position &&
                position.x == mousedown_position.x &&
                position.y == mousedown_position.y)
            {
                move_to(position.x, position.y);
                if (modes.DRAG    == mode) drag_cancel();
                if (modes.GESTURE == mode) path_cancel();
            }
            mousedown_position = undefined;
        }
        switch (mode)
        {
        case modes.DRAG:
            drag_end();
            break;
        case modes.GESTURE:
            path_end();
            break;
        }
        mode = modes.NONE;
    }
    // These values are defined in the DOM3 specification,
    // but it is still a draft and it is not implemented
    // in general apart from Gecko browsers.
    var key_codes = {
        DOM_VK_BACK_SPACE: 0x08,// Not in DOM3
        DOM_VK_TAB:        0x09,// Not in DOM3
        DOM_VK_ENTER:      0x0C,
        DOM_VK_LEFT:       0x25,// Code 0x14 in DOM3
        DOM_VK_RIGHT:      0x27,// Code 0x15 in DOM3
        DOM_VK_UP:         0x26,// Code 0x16 in DOM3
        DOM_VK_DOWN:       0x28,// Code 0x17 in DOM3
        DOM_VK_DELETE:     0x2E // Code 0x0A in DOM3
    };
    var keydown_before = false;
    function keypress(event)
    {
        standard_event(event);
        var key = event.charCode;
        if (key == undefined) key = event.which;
        var consumed = true;
```

```
    if (0x20 == key)                          do_command( "right" );
    else if (0x08 == key && !keydown_before) do_command( "backspace" );
    else if (0xFF == key && !keydown_before) do_command( "delete" );
    else if (0x09 == key && event.shiftKey)  do_command( "backtab" );
    else if (0x09 == key)                     do_command( "tab" );
    else if (!key && event.keyCode
             && !keydown_before) return keydown(event);
    else if (key>0x20 && !(event.ctrlKey || event.metaKey))
    {
        insert(String.fromCharCode(key));
    }
    else if (!keydown_before) consumed = false;
    if (consumed) event.preventDefault();
    keydown_before = false;

    return !consumed;
}
function keydown(event)
{
    standard_event(event);
    var key = event.keyCode;
    if (!event.keyCode && event.which) key = event.which;
    var consumed = true;
    if (event.ctrlKey|event.altKey|event.metaKey)
    {
        consumed = false;
    }
    else switch (key)
    {
    case key_codes.DOM_VK_RIGHT: do_command( "right" ); break;
    case key_codes.DOM_VK_LEFT:  do_command( "left" );  break;
    case key_codes.DOM_VK_UP:    do_command( "up" );    break;
    case key_codes.DOM_VK_DOWN:  do_command( "down" );  break;
    case key_codes.DOM_VK_BACK_SPACE:
        do_command( "backspace" );
        break;
    case key_codes.DOM_VK_DELETE: do_command( "delete" ); break;
    case key_codes.DOM_VK_TAB:
        if (event.shiftKey) do_command( "backtab" );
        else                do_command( "tab" );
        break;
    default: consumed = false;
    }
```

```
        if (consumed) event.preventDefault();
        keydown_before = true;

        return !consumed;
    }
    var tacto_has_focus = false;
    function focus(event)
    {
        tacto_has_focus = true;
        caret_blink(tacto_has_focus);
    }
    function blur(event)
    {
        tacto_has_focus = false;
        caret_blink(tacto_has_focus);
    }

    var touch_initial, touch;
    function touchstart(event)
    {
        if (event.touches.length == 2)
        {
            touch_initial =
                {
                    x: (event.touches.item(0).pageX
                        + event.touches.item(1).pageX)
                        / 2 - viewport.offsetLeft,
                    y: (event.touches.item(0).pageY
                        + event.touches.item(1).pageY)
                        / 2 - viewport.offsetTop,
                };
            touch = { x: touch_initial.x, y: touch_initial.y };
        }
        else if (event.touches.length == 1)
        {
            event.preventDefault();
            mousedown(event);
        }
    }
    function touchmove(event)
    {
        if (event.touches.length == 2)
        {
            if (!touch) touch = { x:0, y:0 };
```

```
            touch.x  =
                (event.touches.item(0).pageX
                 + event.touches.item(1).pageX) / 2
                - viewport.offsetLeft;
            touch.y  =
                (event.touches.item(0).pageY
                 + event.touches.item(1).pageY) / 2
                - viewport.offsetTop;
            if (!touch_initial)
            {
                touch_initial = { x: touch.x, y: touch.y };
            }
        }
        else if (event.touches.length == 1)
        {
            mousemove(event);
            event.preventDefault();
        }
    }
    function touchend(event)
    {
        event.preventDefault();
        mousedown_position = null;
        mouseup(event);
    }
    function gesturestart(event)
    {
    }
    function gesturechange(event)
    {
        //standard_event(event);// Not needed
        if (touch_initial && touch)
        {
            transform(touch_initial.x,
                      touch_initial.y,
                      touch.x - touch_initial.x,
                      touch.y - touch_initial.y,
                      event.scale,
                      event.rotation);
        }
    }
    function gestureend(event)
    {
        //standard_event(event);// Not needed
```

```
        fix();
        touch_initial = undefined;
        touch         = undefined;
    }

    function text_input_change(event)
    {
        standard_event(event);
        var input = event.target;
        if (input.value.length)
        {
            window.setTimeout(function () {
                insert(input.value);
                input.value = "";
            }, 100);
        }
        event.preventDefault();
    }

    function text_input_keydown(input, event)
    {
        if (input.value == "")
        {
            switch (event.keyCode)
            {
            case 8:  // Backspace
            case 46: // Delete
            case 37: // Left
            case 38: // Up
            case 39: // Right
            case 40: // Down
                keydown(event);
            }
        }
    }

    ////////////////////////////////////////////////////////////////
    // Layout analysis



    function BSTNode(argument)
    {
```

```
var self = this;
// public:
this.is_region = is_region;
this.is_symbol = is_symbol;
this.symbol_at = symbol_at;
this.symbols_inside = symbols_inside;

// Constructor:
if (undefined == argument) argument = BSTNode.labels.EXPRESSION;
if (typeof argument == "number")
{
    // Region nodes:
    self.children = [];
    self.parent = null;
    self.label = argument;
    self.symbol_class = null;
    self.node = null;
    self.min_x  = 0;
    self.min_y  = 0;
    self.max_x  = 0;
    self.max_y  = 0;
    self.width  = 0;
    self.height = 0;
    self.centroid_x = 0;
    self.centroid_y = 0;
}
else
{
    // Symbol nodes:
    self.children = {};
    self.parent = null;
    self.label = get_text(argument);
    if ("÷" == self.label || "-" == self.label)
        self.symbol_class = BSTNode.FRACTION_BAR;
    else if (self.label.match(/[0-9]/))
        self.symbol_class = BSTNode.DIGIT;
    else if (self.label.match(/[=<>⋁⋀→⇒+-]/))
        self.symbol_class = BSTNode.NON_SCRIPTED;
    else if (self.label.match(/[\(\{\[]/))
        self.symbol_class = BSTNode.OPEN_BRACKET;
    else if (self.label.match(/[\)\}\]]/))
        self.symbol_class = BSTNode.CLOSE_BRACKET;
    else if (self.label.match(/[√]/))
```

```
            self.symbol_class = BSTNode.ROOT;
        else if (self.label.match( /[ΣʃΠ]/ ))
            self.symbol_class = BSTNode.VARIABLE_RANGE;
        else // We do not need to distinguish ascenders/descenders.
            self.symbol_class = BSTNode.CENTERED;
        self.node = argument;
        var node = self.node;
        self.min_x  = node.offsetLeft;
        self.min_y  = node.offsetTop;
        self.max_x  = node.offsetLeft + node.offsetWidth;
        self.max_y  = node.offsetTop  + node.offsetHeight;
        self.width  = node.offsetWidth;
        self.height = node.offsetHeight;
        self.centroid_x = (self.min_x + self.max_x) / 2;
        self.centroid_y = (self.min_y + self.max_y) / 2;
        for (var i = 0; i < BSTNode.labels.length; ++i)
        {
            self.children[i] = new BSTNode(i);
        }
        if (self.node.hasAttribute( "scaled-width" ))
        {
            self.width = Number(self.node
                                .getAttribute( "scaled-width" ));
            self.min_x = self.centroid_x - self.width / 2;
            self.max_x = self.min_x + self.width;
        }
        if (self.node.hasAttribute( "scaled-height" ))
        {
            self.height = Number(self.node
                                 .getAttribute( "scaled-height" ));
            self.min_y = self.centroid_y - self.height / 2;
            self.max_y = self.min_y + self.height;
        }
    }

    var super_threshold = -0.25;
    var subsc_threshold =  0.25;


    function determine_position(root, node)
    {
        var delta_y = node.centroid_y - root.centroid_y;
        if (root.symbol_class == BSTNode.OPEN_BRACKET)
        {
```

```
            if (root.min_y < node.centroid_y &&
                root.max_y > node.centroid_y)
            {
                if (node.symbol_class == BSTNode.CLOSE_BRACKET)
                {
                    root.symbol_class = BSTNode.BRACKETED;
                }
                return BSTNode.labels.CONTAINS;
            }
        }
        else if (node.centroid_x < root.min_x)
        {
            if (delta_y > subsc_threshold * root.height)
            {
                return BSTNode.labels.BLEFT;
            }
            else if (delta_y < super_threshold * root.height)
            {
                return BSTNode.labels.TLEFT;
            }
            // else it's on the same base line.
        }
        else if (node.centroid_x > root.max_x)
        {
            if (delta_y > subsc_threshold * root.height)
            {
                return BSTNode.labels.SUBSC;
            }
            else if (delta_y < super_threshold * root.height)
            {
                return BSTNode.labels.SUPER;
            }
            // else it's on the same base line.
        }
        else if (root.symbol_class == BSTNode.FRACTION_BAR)
        {
            if (delta_y > 0) return BSTNode.labels.BELOW;
            else             return BSTNode.labels.ABOVE;
        }

        return BSTNode.labels.NULL;
    }
```

```
this.insert = function(node)
{
    if (!node)
    {
        alert( "undefined node in insert()" );
        return false;
    }
    var position, i;
    if (self.is_region())
    {
        i = self.children.length - 1;
        while (i >= 0 && !self.children[i].insert(node)) --i;
        if (i < 0)
        {
            // Was not inserted in any child
            self.add(null, node);
        }
        else
        {
            self.extend(self.children[i]);
        }

        return true;
    }
    else
    {
        position = determine_position(self, node);
        if ((position == BSTNode.labels.SUPER ||
             position == BSTNode.labels.SUBSC) &&
            !node.node.hasAttribute( "stretch" ))
        {
            var style = node.node.style;
            var transition = "transform 0.3s linear";
            style.MozTransition = "-moz-" + transition;
            style.MozTransform = "scale(0.60)";
            style.webkitTransition = "-webkit-" + transition;
            style.webkitTransform = "scale(0.60)";
        }
        if (position != BSTNode.labels.NULL)
        {
            self.children[position].insert(node);

            return true;
```

```
            }
        }

        return false;
    }
    this.extend = function(other)
    {
        if (self.is_region() && 0 == self.children.length)
        {
            self.min_x = other.min_x;
            self.max_x = other.max_x;
            self.min_y = other.min_y;
            self.max_y = other.max_y;
        }
        else
        {
            if (other.min_x < self.min_x) self.min_x = other.min_x;
            if (other.max_x > self.max_x) self.max_x = other.max_x;
            if (other.min_y < self.min_y) self.min_y = other.min_y;
            if (other.max_y > self.max_y) self.max_y = other.max_y;
        }
        self.width  = self.max_x - self.min_x;
        self.height = self.max_y - self.min_y;
        self.centroid_x = (self.min_x + self.max_x) / 2;
        self.centroid_y = (self.min_y + self.max_y) / 2;
    }
    this.merge = function(other)
    {
        if (!self.node) alert( "merge: not node: " + self);
        if (self.is_symbol() && other.is_symbol())
        {
            self.label += other.label;
            for (var c in other.children)
            {
                var region = other.children[c];
                for (var i = 0; i < region.children.length; ++i)
                {
                    self.add(c, region.children[i]);
                }
            }
        }
        self.extend(other);
    }
    this.add = function(position, node)
```

```
{
    var region;
    if (self.is_region())
    {
        region = self;
        // TODO: in the future, use a more capable
        // algorithm for finding the node where
        // this one should be attached,
        // to allow multiple expressions.
        // This is enough if we have only one expression.
        var i, root;
        for (i = 0; i < region.children.length; ++i)
        {
            root = region.children[i];
            if (root.min_x > node.centroid_x) break;
        }
        if (node.is_region())
        {
            var children = node.children;
            region.children = region.children.slice(0, i)
                .concat(children.splice(0, children.length),
                        region.children.slice(i)
                       );
        }
        else
        {
            region.children.splice(i, 0, node);
        }
        region.extend(node);
    }
    else if (self.symbol_class == BSTNode.CLOSE_BRACKET
             && self.parent
             && self.parent.symbol_class == BSTNode.BRACKETED)
    {
        self.parent.add(position, node);
        return;
    }
    else
    {
        region = self.children[position];
        region.add(null, node);
    }
    node.parent = self;
}
```

```
this.toString = function()
{
    var text = "";
    var i;
    if (self.is_region())
    {
        if (self.children.length > 0)
        {
            text += BSTNode.labels[self.label];
            text += "(";
            for (i = 0; i < self.children.length; ++i)
            {
                if (i > 0) text += ", ";
                text += self.children[i];
            }
            text += ") ";
        }
    }
    else
    {
        text += "[" + self.label + "]";
        var children_text = "";
        for (i in self.children)
        {
            children_text += self.children[i];
        }
        if (children_text) text += "{" + children_text + "} ";
    }

    return text;
}

// private:
function is_region() { return (self.node == null); }
function is_symbol() { return (self.node != null); }
var best_fit;
function symbol_at(position)
{
    best_fit = { symbol:null, distance:Number.MAX_VALUE };
    find_best_fit(self, position, best_fit);

    return best_fit.symbol;
}
function find_best_fit(node, position, best_fit)
```

```
{
    var i;
    if (node.is_region())
    {
        for (i = 0; i < node.children.length; ++i)
        {
            find_best_fit(node.children[i],
                          position, best_fit);
        }
    }
    else
    {
        if (position.x >= node.min_x &&
            position.x <= node.max_x &&
            position.y >= node.min_y &&
            position.y <= node.max_y)
        {
            var minkowski_distance =
                Math.abs(node.centroid_x - position.x) +
                Math.abs(node.centroid_y - position.y);
            if (minkowski_distance < best_fit.distance)
            {
                best_fit.symbol   = node;
                best_fit.distance = minkowski_distance;
            }
        }

        for (i in node.children)
        {
            find_best_fit(node.children[i],
                          position, best_fit);
        }
    }
}
function symbols_inside(path)
{
    return collect_symbols_inside(path, self, []);
}
function collect_symbols_inside(path, node, nodes)
{
    var i;
    if (node.is_region())
    {
        for (i = 0; i < node.children.length; ++i)
```

```
            {
                collect_symbols_inside(path, node.children[i],
                                       nodes);
            }
        }
        else
        {
            if (path.contains(node.centroid_x, node.centroid_y))
            {
                nodes.push(node);
            }

            for (i in node.children)
            {
                collect_symbols_inside(path, node.children[i],
                                       nodes);
            }
        }

        return nodes;
    }
}
BSTNode.labels = ["NULL", "ABOVE", "BELOW", "SUPER", "SUBSC",
                  "UPPER", "LOWER", "TLEFT", "BLEFT",
                  "CONTAINS", "EXPRESSION"];
for (var i = 0; i < BSTNode.labels.length; ++i)
{
    BSTNode.labels[BSTNode.labels[i]] = i;
}
BSTNode.FRACTION_BAR   = 1;
BSTNode.DIGIT          = 2;
BSTNode.NON_SCRIPTED   = 3;
BSTNode.OPEN_BRACKET   = 4;
BSTNode.ROOT           = 5;
BSTNode.VARIABLE_RANGE = 6;
BSTNode.CENTERED       = 7;
// Additional node types not defined in DRACULAE:
BSTNode.INTEGER        = 8;
BSTNode.REAL           = 9;
BSTNode.FUNCTION       = 10;
BSTNode.CLOSE_BRACKET  = 11;
BSTNode.BRACKETED      = 12

function get_symbols()
```

```
{
    var nodes = [];
    var node = formula.firstChild;
    while (node)
    {
        if (is_item(node)) nodes.push(new BSTNode(node));
        node = node.nextSibling;
    }

    return nodes;
}

var tree;
function update()
{
    tree = new BSTNode();
    var symbols = get_symbols();
    symbols.sort(function (a,b) { return(a.min_x-b.min_x); });
    var i;
    for (i = 0; i < symbols.length; ++i)
    {
        if (!symbols[i].node.hasAttribute("stretch"))
        {
            symbols[i].node.style.MozTransform = "scale(1.0)";
            symbols[i].node.style.webkitTransform = "scale(1.0)";
        }
        tree.insert(symbols[i]);
    }
}

function get_expression(formats)
{
    if (!tree) update();

    var om, english_text;
    var expression = {};
    for (var i = 0; i < formats.length; ++i)
    {
        var format = formats[i];
        switch (format)
        {
        case "pmml":
            try       { expression[format] = to_MathML(tree); }
```

```
                catch (e) { expression[format] = e.toString();    }
                alert( "mathml: " + (new XMLSerializer()).serializeToString(to_MathML(tr
                break;
            case "openmath":
                try       { if (!om) om = to_OpenMath(tree); }
                catch (e) { expression[format] = String(e);  }
                if (om) expression[format] = om;
                break;
            case "bst":
                try       { expression[format] = tree;         }
                catch (e) { expression[format] = e.toString(); }
                break;
            case "text":
                try
                {
                    expression[format] = [];
                    if (!om) om = to_OpenMath(tree);
                    for (var n = 0; n < om.length; ++n)
                    {
                        english_text = om_to_english(om[n]);
                        expression[format].push(english_text);
                    }
                }
                catch (e) { expression[format] = e.toString(); }
                break;
            case "voice":
                try
                {
                    if (!om) om = to_OpenMath(tree);
                    if (!english_text) english_text = om_to_english(om[0]);
                    expression[format] = new Voice();
                    expression.voice.prepare(english_text);
                }
                catch (e) { expression[format] = e.toString(); }
            };
        }

        update();// Restore tree after combining symbols.

        return expression;
    }
```

```
function combine_symbols(symbols)
{
    var i = 0;
    while (i < symbols.length)
    {
        var previous, next;
        var child = symbols[i];
        if (BSTNode.DIGIT == child.symbol_class
            && previous
            && (previous.symbol_class == BSTNode.DIGIT
                || previous.symbol_class == BSTNode.INTEGER
                || previous.symbol_class == BSTNode.REAL)
           )
        {
            previous.merge(child);
            if (BSTNode.DIGIT == previous.symbol_class)
            {
                previous.symbol_class = BSTNode.INTEGER;
            }
            symbols.splice(i, 1);
        }
        else if ((","  == child.label ||  "."  == child.label)
                 && previous
                 && (previous.symbol_class == BSTNode.DIGIT
                     || previous.symbol_class == BSTNode.INTEGER)
                 && i+1 < symbols.length && (next = symbols[i+1])
                 && next.symbol_class == BSTNode.DIGIT
                 && next.min_x - child.max_x < previous.width)
        {
            previous.merge(child);
            previous.symbol_class = BSTNode.REAL;
            symbols.splice(i, 1);
        }
        else
        {
            previous = child;
            ++i;
        }
    }
}

////////////////////////////////////////////////////////////////
// MathML translation
```

```
var ns_MathML = "http://www.w3.org/1998/Math/MathML";
function create_mathml_element(name)
{
    return document.createElementNS(ns_MathML, name);
}
function to_MathML(bst_node)
{
    var nodes;
    var i;
    if (!bst_node)
    {
        nodes.push(create_mathml_element("merror"));
        set_text(nodes[nodes.length-1], "no bst_node");
    }
    else if (bst_node.is_region())
    {
        var math_nodes = [], last_math_node;
        var previous, digits;
        for (i = 0; i < bst_node.children.length; ++i)
        {
            var child = bst_node.children[i];
            if (("," == child.label || "." == child.label)
                && previous
                && previous.symbol_class == BSTNode.DIGIT)
            {
                last_math_node = math_nodes[math_nodes.length-1];
                if (last_math_node.nodeName == "mn")
                {
                    append_text(last_math_node, ".");
                    continue;// Do not update previous
                }
            }
            else if (previous
                     && previous.symbol_class == BSTNode.DIGIT
                     && previous.symbol_class == child.symbol_class
                    )
            {
                last_math_node = math_nodes[math_nodes.length-1];
                if (last_math_node.nodeName == "mn")
                {
                    append_text(last_math_node, child.label);
                }
            }
            else
```

```
            {
                math_nodes = math_nodes.concat
                (to_MathML(bst_node.children[i]));
            }
            previous = child;
        }
        if (BSTNode.labels.EXPRESSION == bst_node.label)
        {
            var math = create_mathml_element( "math" );
            for (i = 0; i < math_nodes.length; ++i)
            {
                math.appendChild(math_nodes[i]);
            }
            nodes = [math];
        }
        else
        {
            nodes = math_nodes;
        }
    }
    else if (bst_node.is_symbol())
    {
        var mathml;
        switch (bst_node.symbol_class)
        {
        case BSTNode.DIGIT:
            mathml = create_mathml_element( "mn" );
            break;
        case BSTNode.CENTERED:
            mathml = create_mathml_element( "mi" );
            break;
        case BSTNode.FRACTION_BAR:
        case BSTNode.NON_SCRIPTED:
        case BSTNode.ROOT:
        case BSTNode.VARIABLE_RANGE:
            mathml = create_mathml_element( "mo" );
            break;
        case BSTNode.BRACKETED:
            mathml = create_mathml_element( "mfenced" );
            break;
        case BSTNode.OPEN_BRACKET:
        case BSTNode.CLOSE_BRACKET:
            return [];
            break;
```

```
default:
    alert( "Error: unhandled symbol class "
          + bst_node.symbol_class
          + " for " + bst_node.label);
    return nodes;
}
set_text(mathml, bst_node.label);
var r_contains, r_super, r_subsc, r_above, r_below;
r_contains = bst_node.children[BSTNode.labels.CONTAINS];
r_super = bst_node.children[BSTNode.labels.SUPER];
r_subsc = bst_node.children[BSTNode.labels.SUBSC];
r_above = bst_node.children[BSTNode.labels.ABOVE];
r_below = bst_node.children[BSTNode.labels.BELOW];
if (bst_node.symbol_class == BSTNode.BRACKETED
    && r_contains && r_contains.children.length)
{
    set_text(mathml, "" );
    mathml.appendChild(mrow(to_MathML(r_contains)));
}
if (bst_node.symbol_class == BSTNode.FRACTION_BAR
    && r_above && r_above.children.length
    && r_below && r_below.children.length)
{
    var mfrac = create_mathml_element( "mfrac" );
    mfrac.appendChild(mrow(to_MathML(r_above)));
    mfrac.appendChild(mrow(to_MathML(r_below)));
    mathml = mfrac;
}
if (r_super && r_super.children.length)
{
    var msup = create_mathml_element( "msup" );
    msup.appendChild(mathml);
    msup.appendChild(mrow(to_MathML(r_super)));
    mathml = msup;
}
if (r_subsc && r_subsc.children.length)
{
    var msub = create_mathml_element( "msub" );
    msub.appendChild(mathml);
    msub.appendChild(mrow(to_MathML(r_subsc)));
    mathml = msub;
}
if (mathml) nodes = [mathml];
else        nodes = [];
```

```
        }
        else
        {
            // Error
            alert( "to_MathML(): Unkown bst_node type: " + bst_node);
        }

        return nodes;
    }
    function mrow(nodes)
    {
        var mrow;
        if (1 == nodes.length)
        {
            mrow = nodes[0];
        }
        else
        {
            mrow = create_mathml_element( "mrow" );
            for (var i = 0; i < nodes.length; ++i)
            {
                mrow.appendChild(nodes[i]);
            }
        }

        return mrow;
    }

    ////////////////////////////////////////////////////////////////
    // OpenMath translation

    var ns_OpenMath = "http://www.openmath.org/OpenMath";
    var openmath_symbols =
        {
            "+" : [ "arith1" , "plus" ], "-" : [ "arith1" , "minus" ],
            "×" : [ "arith1" , "times" ], "·" : [ "arith1" , "times" ],
            "÷" : [ "arith1" , "divide" ], "/" : [ "arith1" , "divide" ],
            "-" : [ "arith1" , "divide" ],
            "=" : [ "relation1" , "eq" ],
            ">" : [ "relation1" , "gt" ], "<" : [ "relation1" , "lt" ],
            "!" : [ "nums1" , "factorial" ],
            "π" : [ "nums1" , "pi" ]
        };
```

```
var operators =
    {
        // cd:name [precedence, arity_postfix, arity_prefix, right_assoc]
        "arith1":
        {"plus" : [10, 1, 1, false],  "minus": [10, 1, 1, false],
         "times": [20, 1, 1, false], "divide": [20, 1, 1, false]},
        "nums1":
        {"factorial": [30, 1, 0, true]},
        "relation1":
        {"eq": [5, 1, 1, false],
         "gt": [5, 1, 1, false], "lt": [5, 1, 1, false]},
        "matracas.org":
        {"mixed-number": [100, 1, 1, false]}
    };
function OM(element_name)
{
    return document.createElementNS(ns_OpenMath, element_name);
}
function OMI(digits)
{
    var node = OM("OMI");
    node.appendChild(document.createTextNode(digits));
    return node;
}
function OMF(digits)
{
    var node = OM("OMF");
    node.setAttribute("dec", digits);
    return node;
}
function OMV(name)
{
    var node = OM("OMV");
    node.setAttribute("name", name);
    return node;
}
function OMS(cd, name)
{
    var node = OM("OMS");
    node.setAttribute("cd",   cd);
    node.setAttribute("name", name);
    return node;
}
```

```
function om_cd(node)   { return node.getAttribute( "cd" );   }
function om_name(node) { return node.getAttribute( "name" ); }

function OMSTRING(text)
{
    var node = OM( "OMSTRING" );
    node.appendChild(document.createTextNode(text));
    return node;
}
function OME(description)
{
    var node = OM( "OME" );
    node.appendChild(OMS( "error" , "runtime" ));
    node.appendChild(OMSTRING(description));
    return node;
}


function parse_operators(math_nodes)
{
    var operator_stack = [];
    var argument_stack = [];
    var last_operator_precedence = 0;
    var last_node_was_argument = false;
    var i, j;
    i = 0;
    while (i <= math_nodes.length)
    {
        var node, op;
        op = null;
        if (i < math_nodes.length)
        {
            node = math_nodes[i];
            if ( "OMS" == node.localName)
            {
                op = operators[om_cd(node)][om_name(node)];
            }
        }
        else
        {
            node = null;
            op = [0,0,0,false];// EOF
        }
        if (!op && last_node_was_argument)
```

```
{
    // This is the operator when two objects are juxtaposed.
    if (math_nodes[i-1].localName == "OMI"
        && node.localName == "OMA" && node.firstChild
        && "nums1"     == om_cd(node.firstChild)
        && "rational" == om_name(node.firstChild))
    {
        node = OMS("matracas.org", "mixed-number");
        op = operators["matracas.org"]["mixed-number"];
        --i;
    }
    else
    {
        node = OMS("arith1", "times");
        op = operators["arith1"]["times"];
        --i;
    }
}
++i;

if (!op)
{
    last_node_was_argument = true;
    argument_stack.push(node);
}
while (op)
{
    last_node_was_argument = false;
    var application;
    var right_assoc = op[3];
    if (op[0] > last_operator_precedence
        || (op[0] == last_operator_precedence
            && right_assoc)
        || operator_stack.length < 1)
    {
        // Shift:
        if (node) operator_stack.push({"node":node,
                                       "op":op});
        last_operator_precedence = op[0];
        op = null;
    }
    else
    {
```

```
                    // Reduce:
                    var last = operator_stack.pop();
                    application = OM( "OMA" );
                    application.appendChild(last.node);
                    var begin = (argument_stack.length
                                 - last.op[2] - last.op[1]);
                    if (begin < 0) begin = 0;// Error: missing arguments
                    for (j = begin; j < argument_stack.length; ++j)
                    {
                        application.appendChild(argument_stack[j]);
                    }
                    argument_stack
                        .splice(begin,
                                argument_stack.length - begin,
                                application);
                    if (operator_stack.length > 0)
                    {
                        last = operator_stack[operator_stack.length - 1];
                        last_operator_precedence = last.op[0];
                    }
                    else
                    {
                        last_operator_precedence = 0;
                    }
                }
            }
        }

        return argument_stack;
    }
    function to_OpenMath(bst_node)
    {
        var nodes;
        var i;
        if (!bst_node)
        {
            nodes = [OME( "no bst_node" )];
        }
        else if (bst_node.is_region())
        {
            var math_nodes = [];
            combine_symbols(bst_node.children);
            for (i = 0; i < bst_node.children.length; ++i)
            {
```

```
            var child = bst_node.children[i];
            math_nodes = math_nodes.concat(to_OpenMath(child));
        }
        try
        {
            math_nodes = parse_operators(math_nodes, omobj);
        }
        catch (e)
        {
            // TODO: better error reporting
            math_nodes.push(OME("Syntax error: " + e));
        }
        if (BSTNode.labels.EXPRESSION == bst_node.label)
        {
            var omobj = OM("OMOBJ");
            for (i = 0; i < math_nodes.length; ++i)
            {
                omobj.appendChild(math_nodes[i]);
            }
            nodes = [omobj];
        }
        else
        {
            nodes = math_nodes;
        }
    }
    else if (bst_node.is_symbol())
    {
        var om, om_cd_name;
        switch (bst_node.symbol_class)
        {
        case BSTNode.DIGIT:
        case BSTNode.INTEGER:
            om = OMI(bst_node.label);
            break;
        case BSTNode.REAL:
            om = OMF(bst_node.label.replace(/[,]/, "."));
            break;
        case BSTNode.CENTERED:
            om_cd_name = openmath_symbols[bst_node.label];
            if (om_cd_name) om = OMS(om_cd_name[0], om_cd_name[1]);
            else            om = OMV(bst_node.label);
            break;
        case BSTNode.FRACTION_BAR:
```

```
case BSTNode.NON_SCRIPTED:
case BSTNode.ROOT:
case BSTNode.VARIABLE_RANGE:
case BSTNode.FUNCTION:
    om_cd_name = openmath_symbols[bst_node.label];
    if (om_cd_name) om = OMS(om_cd_name[0], om_cd_name[1]);
    else            om = OMS("", bst_node.label);
    break;
case BSTNode.BRACKETED:
    om = null;
    break;
case BSTNode.OPEN_BRACKET:
case BSTNode.CLOSE_BRACKET:
    return [];
    break;
default:
    alert("Error: unhandled symbol class "
          + bst_node.symbol_class
          + " for " + bst_node.label);
    // At this point, nodes is undefined. Return that.
    return nodes;
}
var r_contains, r_super, r_subsc, r_above, r_below;
r_contains = bst_node.children[BSTNode.labels.CONTAINS];
r_super = bst_node.children[BSTNode.labels.SUPER];
r_subsc = bst_node.children[BSTNode.labels.SUBSC];
r_above = bst_node.children[BSTNode.labels.ABOVE];
r_below = bst_node.children[BSTNode.labels.BELOW];
var application, argument;
if (bst_node.symbol_class == BSTNode.BRACKETED
    && r_contains && r_contains.children.length)
{
    om = to_OpenMath(r_contains)[0];
}
if (r_subsc && r_subsc.children.length)
{
    application = OM("OMA");
    application.appendChild(OMS("algebra1",
                                "vector_selector"));
    application.appendChild(om);
    application.appendChild(to_OpenMath(r_subsc)[0]);
    om = application;
}
```

```
        if (bst_node.symbol_class == BSTNode.FRACTION_BAR
            && r_above && r_above.children.length
            && r_below && r_below.children.length)
        {
            var numerator, denominator;
            numerator   = to_OpenMath(r_above)[0];
            denominator = to_OpenMath(r_below)[0];
            if (numerator.localName == "OMI"
                && denominator.localName == "OMI")
            {
                application = OM("OMA");
                application.appendChild(OMS("nums1",
                                            "rational"));
                application.appendChild(numerator);
                application.appendChild(denominator);
            }
            else
            {
                application = OM("OMA");
                application.appendChild(om);
                application.appendChild(numerator);
                application.appendChild(denominator);
            }
            om = application;
        }
        if (r_super && r_super.children.length)
        {
            application = OM("OMA");
            application.appendChild(OMS("arith1", "power"));
            application.appendChild(om);
            application.appendChild(to_OpenMath(r_super)[0]);
            om = application;
        }
        if (om) nodes = [om];
        else    nodes = [];
    }
    else
    {
        // Error
        alert("to_OpenMath(): Unkown bst_node type: " + bst_node);
        nodes = [];
    }

    return nodes;
```

```
}


// Reference materials:
// Mathematical Markup Language (MathML) Version 2.0
// http://www.w3.org/TR/MathML/
// Gecko DOM Reference:
// https://developer.mozilla.org/en/Gecko_DOM_Reference
// TouchEvent Class Reference:
// http://developer.apple.com/library/safari/#documentation/
//        UserExperience/Reference/TouchEventClassReference/
//        TouchEvent/TouchEvent.html


// Constructor:
insert_handles();

viewport.style.position = "relative";
formula.setAttribute("tabindex", "0");

formula.style.position = "absolute";
formula.style.left = "0";
formula.style.right = "0";
formula.style.top = "0";
formula.style.bottom = "0";
disable_selection(formula);
standard_addEventListener(viewport);
standard_addEventListener(formula);
viewport.addEventListener("mousedown",     mousedown,     false);
viewport.addEventListener("mousemove",     mousemove,     false);
viewport.addEventListener("mouseup",       mouseup,       false);
viewport.addEventListener("keypress",      keypress,      false);
viewport.addEventListener("keydown",       keydown,       false);
formula.addEventListener("focus",         focus,         false);
formula.addEventListener("blur",          blur,          false);
viewport.addEventListener("touchstart",    touchstart,    false);
viewport.addEventListener("touchmove",     touchmove,     false);
viewport.addEventListener("touchend",      touchend,      false);
viewport.addEventListener("gesturestart",  gesturestart,  false);
viewport.addEventListener("gesturechange", gesturechange, false);
viewport.addEventListener("gestureend",    gestureend,    false);
viewport.addEventListener("focus",
```

```
                              function () { formula.focus(); },
                              false);
    viewport.addEventListener("blur",
                              function () { formula.blur();  },
                              false);
    formula.style.cursor = "text";
    caret_blink(false);
}
```