Ahmad Akra

Computer algebra system for phones and tablets

Computer Science Tripos, Part II

Trinity College

April 12, 2012

Proforma

Name:	Ahmad Akra
College:	Trinity
Project Title:	CAS for phones and tablets
Examination:	CST, Part II
Year:	2012
Word Count:	10,530
Project Originator:	Dr. Arthur C. Norman
Supervisor:	Dr. Arthur C. Norman

Original aims of the project

To create a computer algebra system (CAS) for Android devices that:

1. Surpasses anything available today for Android in terms of algebra.
2. Displays input and output in natural math notation.
3. Easy to use on a small touch screen.

To achieve this, the aim was to:

1. Port the open source algebra system REDUCE to run on Android,
2. Build on top of it a user interface featuring a math keyboard and LaTeX-style natural math typesetting, hence hiding the textual nature of REDUCE.

Work completed

All the requirements for the project were satisfied, in particular:

1. REDUCE was successfully ported to Android.
2. A natural math typesetting library for Android was implemented which was inspired by the math typesetting of LaTeX.
3. A parser for LaTeX strings was implemented.
4. A user interface was constructed which takes input through a math keyboard and displays both input and output formulae in natural math notation.

Special difficulties

None.

Declaration of originality

I, Ahmad Akra of Trinity College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed

Date

Contents

Chapter 1 Introduction

1.1 Introduction

1.1.1 The project

The goal of this project is to build a Computer Algebra System (CAS) for Android phones and tablets, with an interface that works by touch. The resulting application should give its user access to the features of an algebra system on the phone and should resemble a powerful algebra calculator.

1.1.2 What is an algebra system?

Unlike scientific calculators, an algebra system is a computer program that is capable of performing **symbolic manipulations** such as:

- Expanding
- Refactoring
- Solving equations
- Differentiation
- Integration
- Complex numbers
- Matrix manipulation

Some algebra systems go beyond that and provide extra facilities for plotting graphs, logic and programming.

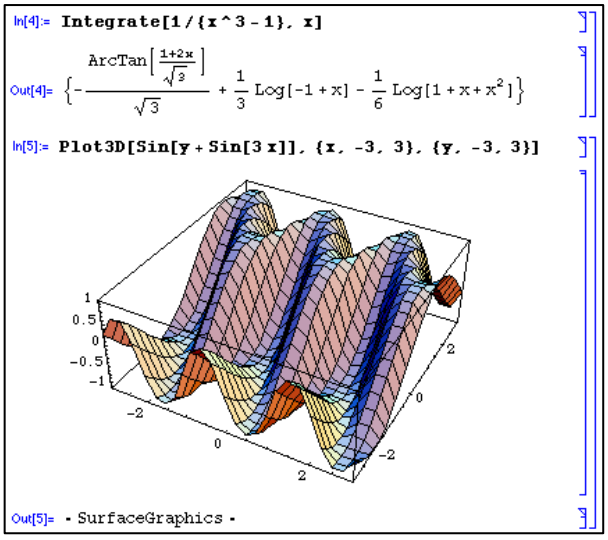


Figure 1: Mathematica is the archetypal example of a commercial algebra system

1.2 The motivation behind this project

As a Cambridge university student, I own a Casio calculator fx-991ES[1] and use it to help me do my homework. This calculator can perform a wide range

[1] http://support.casio.com/pdf/004/fx-115ES_991ES_E.pdf

of calculations and algebra for me despite having a modest processor. Also, it displays input and output in natural math notation despite having a small 31 by 96 pixel screen.
I also happen to own a Galaxy Nexus[2] smart phone running a 1.2 GHz dual core processor, and featuring a 1280 by 720 pixel display. The Android OS on this phone gives me access to over 400,000 apps on the official app store **Google Play**. However, despite all these impressive numbers, I am unable to find a decent algebra system for my Android phone. One, which when compared to the capabilities of the Casio calculator mentioned earlier will give a consistent ratio with that of the computational power of modern smartphones compared to that of the calculator. This is a problem that requires some investigation.
The same problem can be observed more clearly if we compare the leading algebra systems available today for Android to those available for PCs. As I will illustrate in the evaluation section of this dissertation, there is no algebra system for Android phones that comes anywhere near the algebra systems available for PCs (such as Mathematica or Maple) even though modern Android handsets are nearing the benchmarks for PCs and are certainly on par with small netbooks. And even though it seems natural that Android users (and especially students) would like to have an algebra calculator on their phones to assist them in their daily work.
The aim of this project is to address exactly this problem. This project introduces a computer algebra system (CAS) for Android phones which is free, easy to use and, in terms of algebra capabilities, can take advantage of the power of Android handsets.

1.3 The original idea

The idea behind this project is due to Dr. Arthur C. Norman (my project supervisor). Dr. Norman identified the problem that the Android platform lacks a decent CAS. And saw a potential solution to that in the open source programs REDUCE and JLisp in addition to some effort from a final year student.

[2] http://www.google.com/nexus/

1.4 Introducing REDUCE

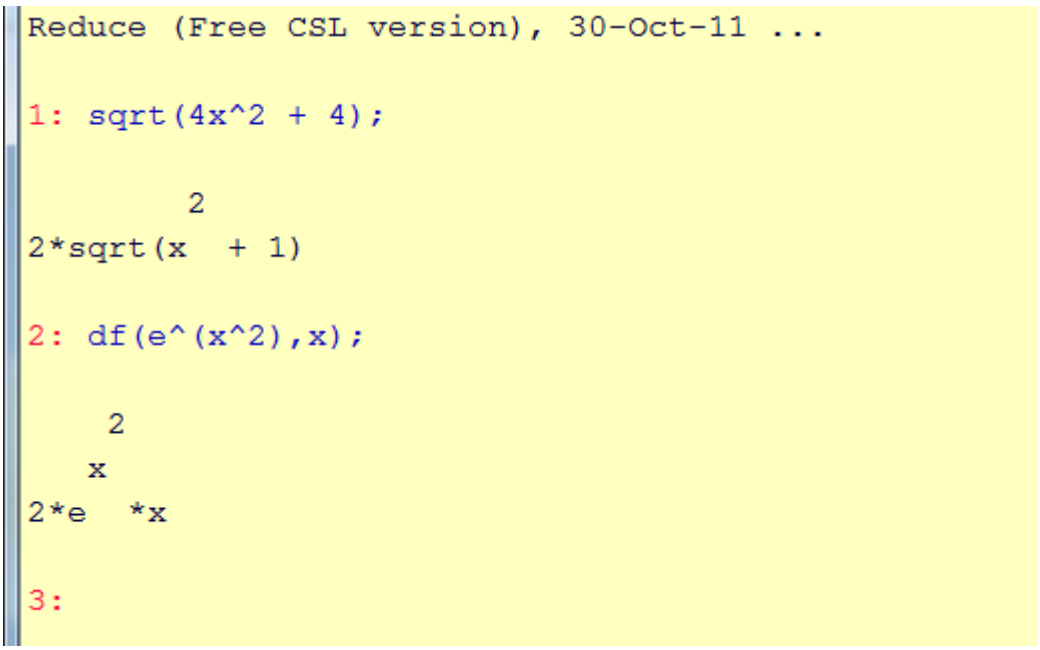


Figure 2: A snapshot of the desktop interface of REDUCE running under JLisp

REDUCE[3] is a large computer algebra system written in a special dialect of Lisp called **Standard Lisp**. It was started by Tony Hearn in 1966, and was made open source in 2008. Over a period of 40 years, and through the contributions of many programmers (including Dr. Norman himself) the core of this algebra system grew to over hundreds of thousands of lines of code, and came to include a wide range of capabilities in algebra, calculus, matrices, series expansions, limits and many more. In terms of size and algebra features, REDUCE is around the same order of magnitude as leading algebra systems such as Mathematica, Maple and Maixma.
REDUCE can be installed on all major operating systems, and can be downloaded as source code from SourceForge repositories[4].

1.5 Introducing JLisp

The instance of REDUCE in Figure 2 was running under the Lisp system **JLisp**. JLisp is an implementation of Standard Lisp written in Java that is capable of compiling Standard Lisp source files into bytecode, and is also capable of executing the compiled bytecode.

[3] http://reduce-algebra.com/
[4] http://reduce-algebra.svn.sourceforge.net/svnroot/reduce-algebra/

To put it in a different way, is a Lisp interpreter written in Java. It can compile and execute Lisp within the Java environment. JLisp was originally created by Dr. Arthur Norman in 1998 to support REDUCE on the PC and to research the feasibility of the Java language for such a task.
JLisp is crucial for this project since it will help port the REDUCE algebra system to the Android platform. REDUCE is written in Lisp, which is not supported on Android. However since JLisp – as described earlier - is Java code that can execute Lisp code, it can potentially solve the problem in the most straightforward way. And enable the Lisp source code of REDUCE to run on the Android JVM.

1.6 My work

My work for this project is to accomplish 3 goals.

1.6.1 Porting REDUCE to Android using JLisp

Although JLisp is Java software, it is not written with Android constraints in mind. Therefore running JLisp on Android requires some modifications to JLisp. For example, JLisp uses API that is not available on Android such as for GUI and for reading files.

1.6.2 Creating the math typesetting library

Which will be used to hide the textual nature of REDUCE under natural looking mathematics. For example, instead of asking users to input something as awkward as `(1/2)*sqrt(x^2-1)`, we ask them to input $\frac{1}{2}\sqrt{x^2-1}$ instead and show them equally nice output. The motivation behind the math typesetting library is the observation that average computer users tend to be repelled by programs with textual interfaces (such as the command line), even if the textual interface is easier to use than the equivalent GUI. Any program with a textual interface is automatically abandoned by a large subset of users and the aim is to avoid this happening to our application.

1.6.3 Building the user interface

The application interface will feature a math keyboard that can fit on the phone screen. It also uses the previously mentioned math typesetting library to show the input and the output in natural math notation.

1.7 Summary

This chapter:

- Introduced the idea of the project and the motivation behind it.
- Introduced two of the key components of the project; REDUCE and JLisp.
- Described the goals that I have to achieve in order to complete this project.

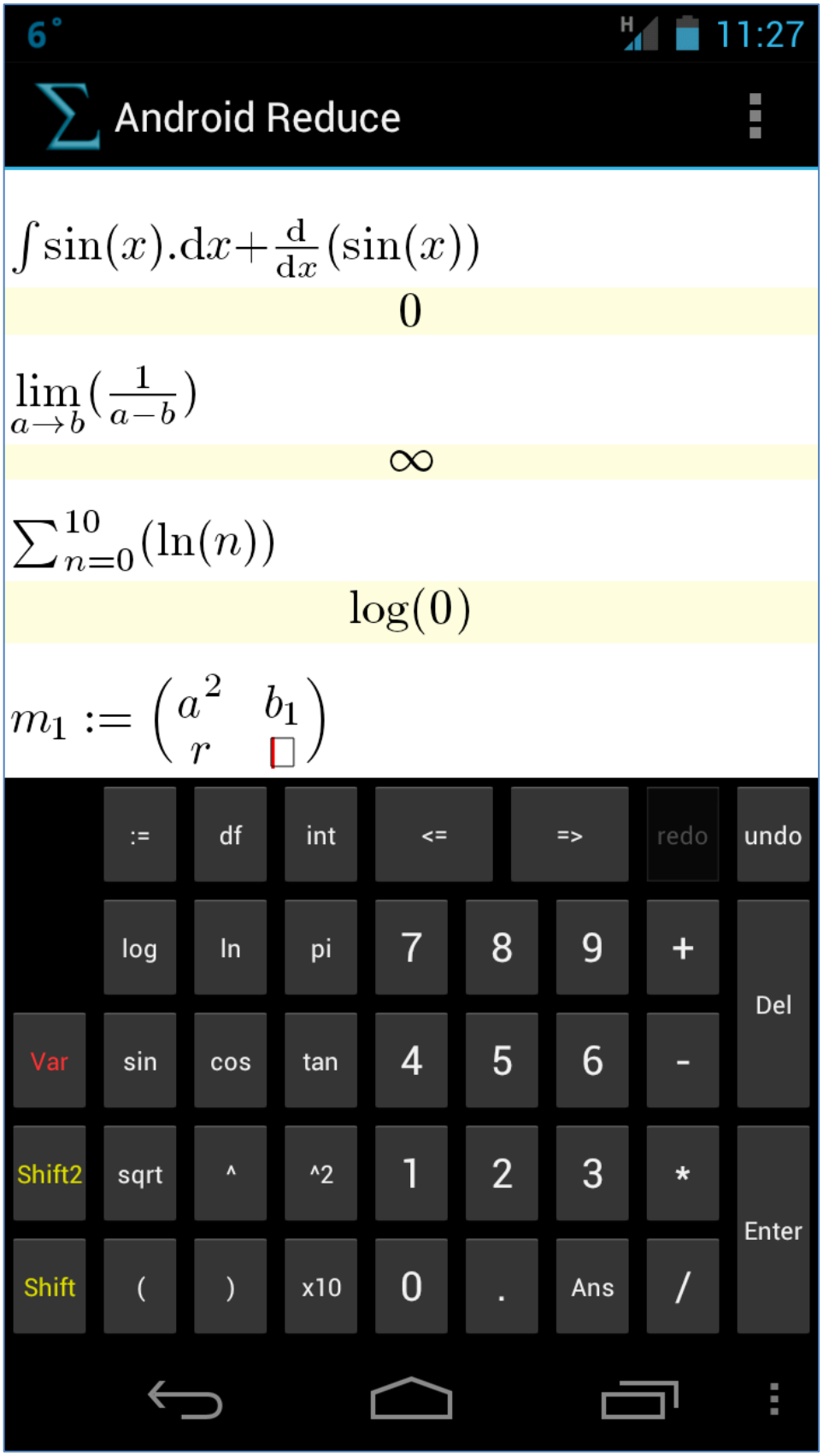


Figure 3: The application interface on Android 4.0 using the Holo theme. More snapshots can be found in Appendix C

Chapter 2 Preparation

2.1 Eliciting requirements

The requirements for this project were specified in the project proposal, under the section "Success Criterion":

1. The REDUCE algebra system should be ported to Android.
2. The output of REDUCE should be arranged to be in true math notation.
3. The input to REDUCE should be possible to enter through a touch UI.

2.1.1 Target users

The target users for this application are **college students**, since they would find it useful, judging by how many of them own and use algebra calculators, also because I am a college student myself and therefore most able to understand their requirements. The design of the application interface will be tailored towards these target users.

2.2 Learning

Before writing any code I had to spend time researching and learning every new software package, algorithm or language that I intended to use but have not used before. Plenty of learning had to be done for this project as will be demonstrated.

2.2.1 Learning REDUCE

The REDUCE source files were obtained from SourceForge. Time was spent in learning about the REDUCE algebra system, in particular, about its algebra capabilities, the semantics and syntax of its input, the nature of its output and the various flags and control parameters that can be used to alter its behaviour. This learning took place in preparation for building the touch interface on top of REDUCE which will hide its textual nature. The REDUCE User's Manual [**1**] proved very useful at this stage.

One convenient feature of the REDUCE algebra system is a flag `fancy` which, if set, causes the algebra system to generate all its output in LaTeX notation.

```
1: on fancy;

2: sin(pi/4);
 latex:\black$\displaystyle  \frac{\sqrt{2}}2$
3:
```

Figure 4: The "on fancy" command switches on the fancy flag which causes REDUCE to generate all output in LaTeX

This feature is available for exactly the reason that I will use it for; to allow developers to extend REDUCE's textual interface and change the output from text mode into natural math notation. The CSL Lisp system[5] (which is similar to JLisp but implemented in C++ instead of Java) makes use of this feature and displays output in natural math notation.

```
5: int(ws,x);  % integrate the result
```

$$x^x$$

```
6: sin(pi/4);
```

$$\frac{\sqrt{2}}{2}$$

Figure 5: The interface of REDUCE running under the CSL Lisp system on the PC.

2.2.2 Learning JLisp

JLisp is key to porting REDUCE to Android, as pointed out in section 1.5 in the introduction.
One thing to consider about JLisp is that it was created in 1998 long before Android existed. Therefore it is not compatible with Android and it uses API that is not available on the Android for GUI and reading and writing to files.
For this project I will make JLisp compatible with Android, and to prepare for that I had to understand JLisp by asking Dr. Norman himself and by reading his research paper on JLisp [**2**].

[5] http://reduce-algebra.svn.sourceforge.net/svnroot/reduce-algebra/trunk/csl/

2.2.3 Learning LaTeX

Even though I never used LaTeX before, this project requires that I learn two distinct aspects of LaTeX:

1. **The LaTeX semantics**

This project involves creating a LaTeX parser to parse the output from REDUCE. Building such a parser requires accurate understanding of the LaTeX semantics for formulae construction.

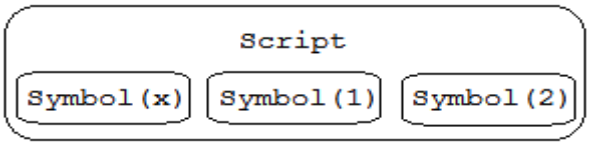
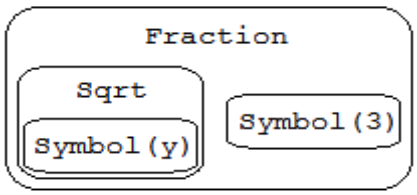
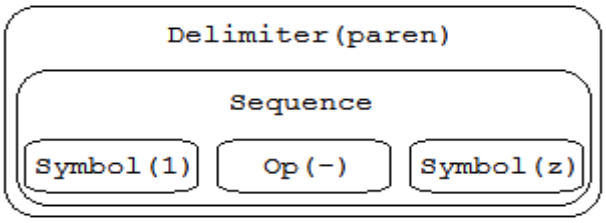
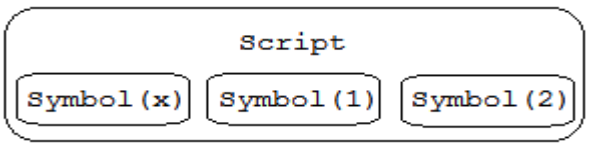
LaTeX string	Parse tree
`x^2_1`	
`\frac{\sqrt{y}}3`	
`\left(1-z\right)`	

Table 1: Examples of LaTeX strings transformed into parse trees

2. **How LaTeX typesets formulae**

One of the project requirements is to build a math typesetting library. To build such a library, I had to learn how this was done before so as not to reinvent the wheel.

Parse tree	Fancy formula
	x_1^2

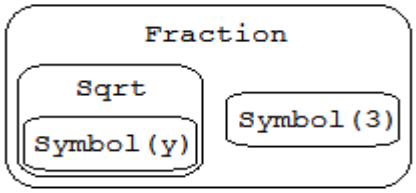
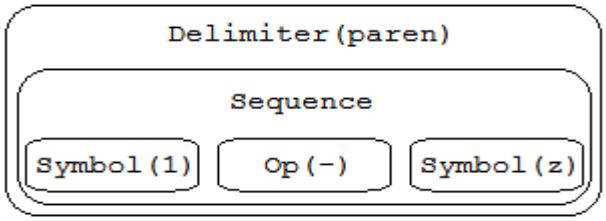
	$\frac{\sqrt{y}}{3}$
	$(1-z)$

Table 2: Examples of how parse trees are typsetted into readible formulae

Since LaTeX formulae are generally considered to be amongst the most beautiful. I decided to make the output of my math typesetting library as close as possible to LaTeX typesetting, and to do that I had to learn how LaTeX typesets formulae by reading the relevant chapters from the TeXbook [**3**].
As the learning has revealed, LaTeX treats every formula as a tree of nested boxes so that each box may contain boxes inside it.

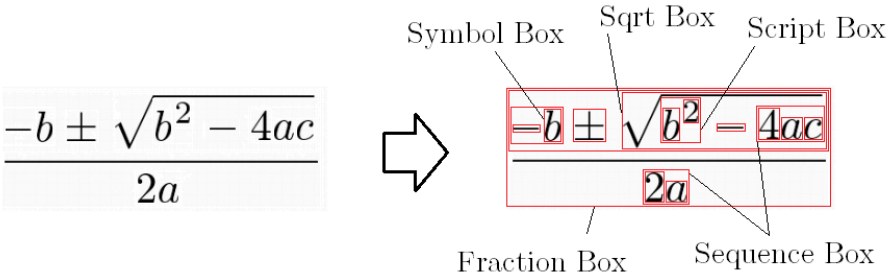


Figure 6: Illustration of how a formula can be treated as a tree of boxes

Every box has a **width**, a **height**, an **axis**, and a **depth**, where height = axis + depth.

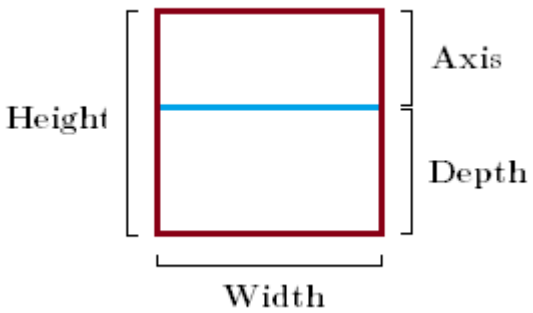


Figure 7: Illustration of the width, height, axis and depth properties

I also learned how every box in the formula is assigned a **style**. There are 8 different styles in LaTeX:

1. **Display**
2. **Text**
3. **Script**
4. **Scriptscript**

And a **cramped** version of every one of these (e.g. cramped script).
The style of a box controls its font size and the internal layout of its contents. LaTeX allows the user to override the style of a certain box. For example, the user can override the style to **display** using the LaTeX command `\displaystyle`.
The styles are assigned to boxes according to a fixed set of rules and tables which I had to learn during the preparation phase and later implement in my math typesetting library. Here is an example table:

If $\sqrt{X}$ is in style...	**Then X is in style...**
display	cramped display
cramped display	cramped display
text	cramped text
cramped text	cramped text
script	cramped script
cramped script	cramped script
scriptscript	cramped scriptscript
cramped scriptscript	cramped scriptscript

Table 3: showing the relationship between the style of the squareroot box and the style of its body

The learning also revealed that LaTeX defines for every **character**, a set of dimension properties in order to position the character correctly relative to other characters within a formula. These properties are:

1. **Bounding height**, which is the height of the character.
2. **Axis height**, the axis lines of the characters aligned when they are positioned next to one another in a string.
3. **Bounding width**: the width of the character.
4. **Advancing width**: the width which the character occupies when in a string (notice that advancing width $\geq$ bounding width)
5. **Italic correction**: (in the context of formula typesetting) specifies how much the superscript of this character should be shifted. For example f has a larger italic correction than a. The superscript of f need to be shifted to the right to make it look better.

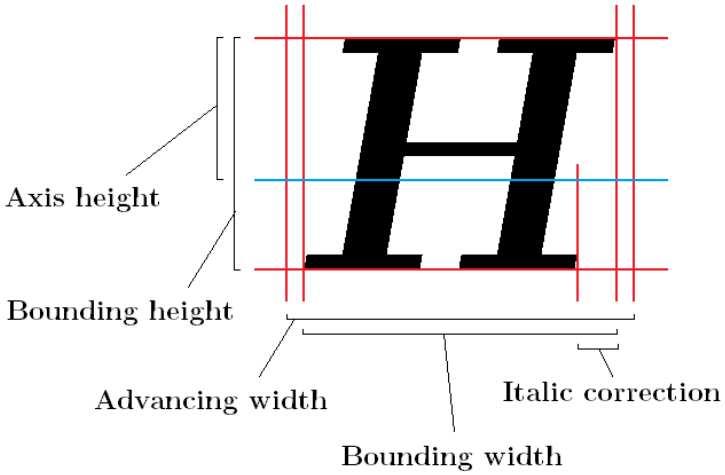


Figure 8: The properties that LaTeX assigns to every character

2.2.4 Learning the Computer Modern fonts

LaTeX typesets a single formula from a family of fonts called the **Computer Modern** family. I had to study these fonts and learn how to use them on Android..

2.2.5 Learning about formula fitting

I researched and gathered ideas on how to approach the problem of math formulae when they do not fit on the narrow screen. I think the paper by Michael Downes titled "Breaking Equations" [**4**] to be quite useful.

2.2.6 Learning Android

Being completely new to Android development and Android all together, I had to spend some preparation time learning the Android mind set and API. The online developer guide [**5**] from Google is an excellent introduction to Android development. I also learned a lot from a video course on Pluralsight[6] titled "Introduction to Android development" [**6**].
One of the things I had to learn is how to use font files on Android. This was essential for creating the math typesetting library. I also had to learn how to build an Android UI composed of activities[7] or screens.

[6] http://www.pluralsight-training.net/
[7] http://developer.android.com/guide/topics/fundamentals/activities.html

2.3 Designing the architecture

After the learning stage, I laid down a plan for the application architecture. This plan will be described with references to Figure 8 below. Understanding this plan will help the reader understand chapter 3 on implementation.

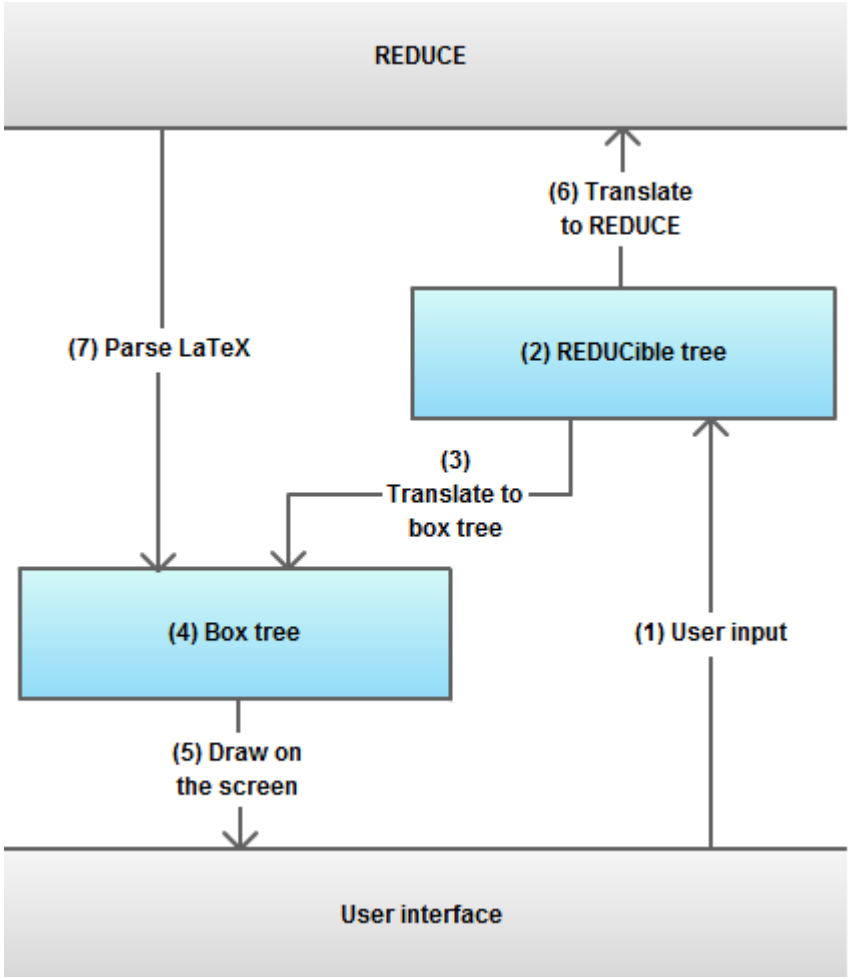


Figure 8: The code architecture plan

I will be building the components between REDUCE and the user interface. The user types a formula for **input** (1), as the user types the input, the user is in reality sending commands for building a structure called the **REDUCible tree** (2), this tree is a logical representation for the formula that the user is typing. While the user is typing the formula and building the REDUCible tree, this tree is constantly translated into a **box tree** (3) (4), which can **draw** itself on the screen (5), this allows the user to see the input formula while typing it. After the user finishes typing the formula and taps enter, the formula is translated to the **REDUCE language** (6) and sent to REDUCE which processes it and prints the output in LaTeX. A special parser **parses that**

LaTeX (7) into a box tree (4) which again can **draw** itself on the screen (5) in order for the user to see the output.

This clean layered architecture provides separation of concern and allows easy development of each part separately. For example, the REDUCible tree does not know anything about the user input and the box tree does not know anything about the REDUCible tree and so on.

2.4 The development process

During the project development I will aim to follow the **agile** development principles published in the agile manifesto in 2001 [**7**]. The principles relevant to this project are:

- Deliver working software frequently.
- Working software is the primary measure of progress.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity is essential.

To follow the 1st principle above, I employed aspects of the **SCRUM** model described in the Book "Agile software development with Scrum" [**8**]. The project code was designed to be incremental such that after every development sprint (2 weeks), the code was in a deliverable state and could be released as a product. For example, in the first sprint, the core of the math typesetting library will be completed (which will be a deliverable product on its own), and in the second sprint, the math typesetting library will be expanded to include everything needed for the application, again a deliverable product, and so on.

To follow the 2nd principle I was constantly using the application on my phone in my daily work, to make sure that it is robust and that it **works** in real life. And also to help me prioritize the features that need to be added and the bugs that need to be removed.

The last two principles were followed through care and understanding that complicated, tightly coupled code and quick "hacks" can only lead to complications in the long run. Everything should be part of a well thought plan.

2.5 Summary

This chapter:

- Listed the 3 requirements of the project.
- Explained in detail all the knowledge that had to be learned before proceeding to the implementation stage.
- Described the layered architecture and the agile model that will be employed during development.

Chapter 3 Implementation

In this chapter I will briefly explain how REDUCE was ported to Android and then I will describe how the following components of the application were implemented:

1. The math typesetting library, aka, the formula boxes.
2. The formula fitting algorithm that can fit formulae on the small screen.
3. The REDUCible classes.
4. The LaTeX parser.
5. And the user interface.

3.1 Porting REDUCE CAS to Android

3.1.1 The REDUCE and JLisp sources

The REDUCE source files were obtained from SourceForge[8]. They include the algebra core of REDUCE which is written in LISP, as well as some lisp systems to support the core. One lisp system, **JLisp**[9], was of special interest to this project because it was one that was written in Java.
JLisp is capable of compiling the LISP core of REDUCE into a binary file `reduce.img`, and also capable of executing `reduce.img` with custom input and output streams. JLisp also features a Swing GUI for input and output.
JLisp is 27K lines of code written with text editors. And for this project, I modified JLisp so that it can run inside an Android project in Eclipse.

3.1.2 Porting JLisp to the Android environment

JLisp often uses libraries that are not available on Android such as Java Swing, it also performs actions which are not allowed on Android such as opening files directly using `FileInputStream`. All such cases were tracked down and modified or deleted. In particular:

1. All the Swing code (2K lines) was found and deleted.

[8] http://reduce-algebra.svn.sourceforge.net/svnroot/reduce-algebra/
[9] http://reduce-algebra.svn.sourceforge.net/svnroot/reduce-algebra/trunk/jlisp/

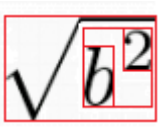
2. The code that reads the `reduce.img` file directly was rewritten to use Android's `AssetManager`, like this:

```
AssetManager mgr = context.getAssets();
final InputStream reduceImgStream = mgr.open("reduce.img");
```

3.2 Implementing the formula boxes

3.2.1 What are the formula boxes?

In a nutshell, the formula boxes like `FractionBox` and `SqrtBox` are the classes that perform the math typesetting. They comprise the math typesetting library which is one of the requirements for this project. They work by building a tree of boxes to represent a formula, for example:



`SqrtBox(ScriptBox(SymbolBox("b"),SymbolBox("2")))`

Every `Box` object in the tree is responsible for:

1. Calculating its own dimensions (width, height and axis) based on the dimensions of its children and providing this information to its parent.
2. Calculating and setting the coordinates of its children based on the LaTeX layout rules. And also drawing any characters that it may contain.

This code snippet illustrates how to typeset the fraction $\frac{1}{2}$ using a tree of boxes:

```
Box numerator, denominator, fraction;

// step 1: construct the tree of boxes
fraction = new FractionBox(
             numerator = new SymbolBox("1"),
             denominator = new SymbolBox("2"));

// step 2: pass a Graphics implementation
fraction.setGraphics(new MyGraphics());

// step 3: call "onDraw" on all boxes, parents before children
fraction.onDraw();
numerator.onDraw();
denominator.onDraw();
```

First a tree of boxes is constructed, then an implementation of the **`Graphics`** interface is passed to the tree (section 3.2.2 talks about this in detail). This interface is used by the boxes to perform the drawing. Then, when the **`onDraw`** method is called on the tree, the formula represented by the tree is typesetted.

Depending on the custom implementation of `Graphics` in the previous code example, it may typeset $\frac{1}{2}$ on anything, for example, a GIF image. My implementation `AndroidGraphics` typesets the formula on an Android layout object and looks like this on the Android screen:

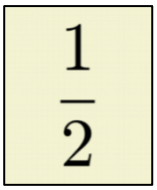


Figure 9: One half rendered by the formula boxes

3.2.2 The `Graphics` interface

This interface is used by the boxes to do the drawing. It contains mainly low level methods that draw characters or lines. Example methods in that interface are:

1. `void drawChar(char c, Font font, int fontNumber, float fontSize, float startX, float startY);`
2. `void drawLine(float lineThickness, float startX, float startY, float stopX, float stopY);`

For example, method number 2 above is used by the boxes to draw the fraction bar, the square root horizontal bar etc...

The Graphics interface is also responsible for supplying size information about characters. For example:

1. `float boundingHeight(char c, Font font, int fontNumber, float fontSize);`
2. `float advancingWidth(char c, Font font, int fontNumber, float fontSize);`
3. `float boundingWidth(char c, Font font, int fontNumber, float fontSize);`

See section 2.2.3 for an explanation of bounding height, advancing width and bounding width.

3.2.3 Implementing the base class `Box`

The base class for all the box types is called `Box`. It is an **abstract** class and it contains the information and behaviour that is common to all boxes. It contains for example:

1. A field for the font size.
2. A field for the LaTeX style (display, text, script etc...).
3. Fields for the box coordinates (explained in section 3.2.5).
4. A reference to the supplied `Graphics` implementation.

The `Box` base class also defines abstract methods, leaving their implementation to the subtypes that inherit from it. For example:

1. `width()`, `height()` and `axis()`, these methods are self-explanatory (see section 2.3.3).
2. `squash()` and `split()`, related to formula fitting (see section 3.3).

3.2.4 The subtypes of `Box` and one example

Each subtype of `Box` represents a formula construct, for example `FractionBox` represents a fraction with a numerator and a denominator. I implemented a range of `Box` subtypes that cover all the basics, these include:: **`SymbolBox`**, **`SpaceBox`**, **`SequenceBox`**, **`FractionBox`**, **`SqrtBox`**, **`DelimiterBox`**, **`ScriptBox`**, **`MultilineBox`** and **`MatrixBox`**.

In this subsection I take one of the simplest subtypes above, `SequenceBox`, as a practical example and work through its implementation. `SequenceBox` is one of the most used box types, it can have a variable number of children and it simply arranges them one after the other in a horizontal sequence such that their axis lines are aligned. For example, the symbols in this formula: $11 + 2 = 13$ are stacked next to one another inside a `SequenceBox`.

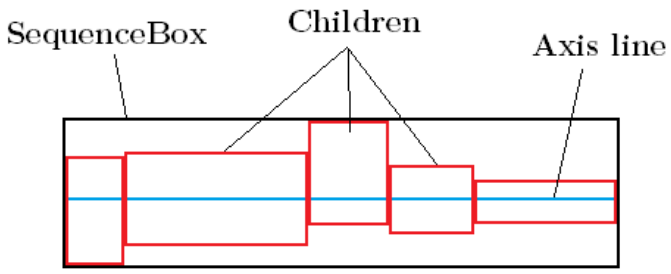


Figure 11: `SequenceBox`, arranges the children in a sequence such that their axis lines are aligned

To implement the `SequenceBox`, the constructor is implemented first. The `SequenceBox` constructor takes a variable number of children as arguments, stores them in a field and sets their style according to the LaTeX rules. In the case of the `SequenceBox`, the style of the children is the same as the style of their parent.

```
public class SequenceBox extends Box {

	Box[] children = new Box[0]; // child boxes

	public SequenceBox(Box... children) {
		if(children != null) this.children = children;
		setChildrenStyle();
	}

	@Override
	public void setChildrenStyle() {
		for(Box box : this.children)
			box.setStyle(style); // “style” field is inherited
	}
```

After the constructor, the dimension methods (width, height and axis) are implemented.

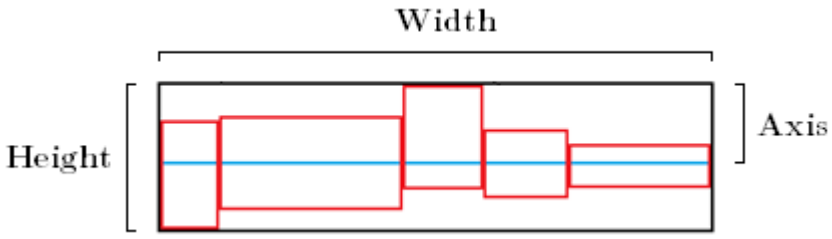


Figure 12: Width, height and axis of a SequenceBox

The **width** of the `SequenceBox` is simply the sum of the widths of its children:

```
	@Override
	public float width() {
		// the sum of all the widths in the sequence
		float width = 0f;
		for(Box box : this.children) {
			width += box.width();
		}
		return width;
	}
```

The **axis** of a box is defined as the distance between the roof of the box and its axis line. From Figure 12: Width, height and axis of a SequenceBox, we can see that the axis of a `SequenceBox` is equal to the maximum axis of any of its children (The middle child in the figure):

```
@Override
public float axis() {
	// the largest axis of any child
	float maxAxis = 0f;
	for(Box box : this.children) {
		maxAxis = Math.max(maxAxis, box.axis());
	}
	return maxAxis;

}
```

Finally the **height** of the `SequenceBox` is the maximum **axis** of any child + the maximum **depth** of any child:

```
@Override
public float height() {
	float maximumAxis = this.axis();

	// the largest depth of any child
	float maximumDepth = 0;
	for(Box box : this.children) {
		maximumDepth = Math.max(maximumDepth, box.depth());
	}
	return maximumAxis + maximumDepth;
}
```

Other `Box` subtypes such as `ScriptBox` have more complicated layout rules than those of `SequenceBox`. All the box subtypes are listed and explained in appendix A.

3.2.5 The `onDraw()` method

Calling the method `onDraw` on the box tree triggers a long sequence of operations. It first calculates the absolute coordinates of every box in the tree, starting from the root all the way to the leaves. And then draws every box according to those absolute coordinates. After `onDraw` returns, there should be a formula displayed to the user on the screen.

3.3 Implementing the formula-fitting algorithm

Since the formulae will be displayed on a phone screen, I devised and implemented an algorithm to deal with the common situation of having a formula that does not fit inside the boundaries of the screen.

```
public static Box fitWithin(Box formula, float maxWidth)
```

This algorithm takes 2 arguments. One argument is a formula (represented by a box tree) and the other argument is a `maxWidth` value. The algorithm attempts to return a new formula that:

1. Looks as nice as possible.
2. Has the same meaning to a human reader as the original.
3. And has a width which is $<$ `maxWidth`.

The algorithm tries to slim the formula down in multiple steps, and after every step, the algorithm checks if the formula width has fallen below the `maxWidth` and if so will stop there and return the result or else it will proceed to the next step. Every step is more effective than the step before but produces a result which is less beautiful and less readable. This section will use the formula below as a running example to illustrate how the implementation of each one of these steps work, and the reader will see that by the 6$^{\text{th}}$ step, the algorithm is able to fit this entire formula in **less than a centimetre width.**

$$\left(E + \frac{e^2}{r}\right)^2 \psi(x) = -\nabla^2\psi(x) + m^2\psi(x)$$

Figure 11: A running example for formula fitting, this figure and all the following figures of this formula were captured from the real application

3.3.1 Step 1: Squash the formula

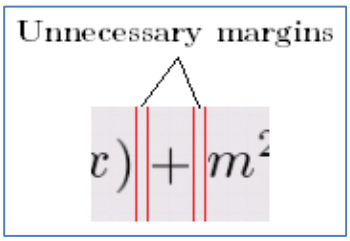


Figure 14: These margins can be removed to make the formula smaller

Many formulae contain empty horizontal margins and spaces that are put there only for aesthetic purposes. If the formula is only slightly wider than `maxWidth`, it may be sufficient to squash those empty margins. The algorithm first asks the box tree (i.e. the formula): How much margin in total, d, can you afford to lose?

```
float d = box.getSquashingDistance(); // recursive method
```

Then the algorithm calculates a value between 0 and 1 called the **`squashingFactor`** which is the fraction of `d` that should **remain,** in order for the formula to have a width of exactly `maxWidth`.

```
float squashingingFactor = Math.max(0, 1-((boxWidth-maxWidth) / d));
```

Here is a visualization to help understand the previous line of code:

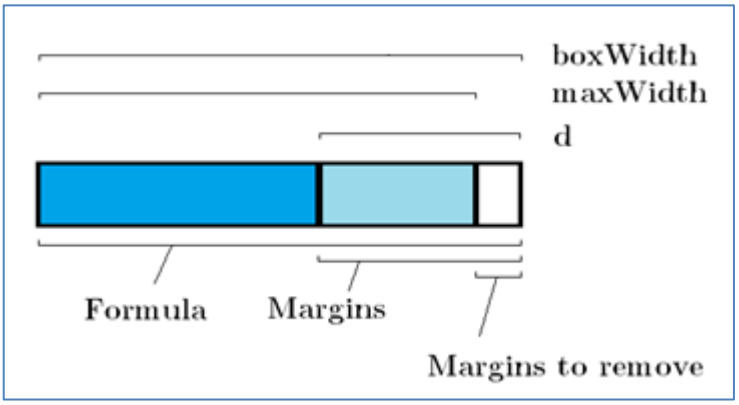


Figure 12: A diagram to help understand how `squashingFactor` is calculated

After the `squashingFactor` is calculated, it is passed to the formula and the formula multiplies each one of its margins by this value.

```
box = box.squash(squashingingFactor);
```

Here is the result if the margins are completely removed (`squashingFactor` = 0):

$$\left(E+\frac{e^2}{r}\right)^2\psi(x)=-\nabla^2\psi(x)+m^2\psi(x)$$

Figure 16: The example formula after removing all the margins

3.3.2 Steps 2-5: Break the formula into multiple lines

If removing margins is not enough to fit the formula, the algorithm proceeds to the next 4 steps: breaking the formula into multiple lines with 4 increasing levels of aggression (These will be explained shortly). The algorithm uses 4 levels of aggression: **easy**, **medium**, **hard** and **nightmare**. In order to break the formula into **lines**, the algorithm splits the formula first into the smallest possible **pieces** or **words**. Then, it arranges those pieces in multiple lines using a word-wrap algorithm.

```
Box[] pieces = box.split(Level.easy);
```

The method `split` is recursive and every subtype of `Box` has a special implementation of `split`. This method performs the splitting into **pieces**. Splitting the boxes into pieces depends on the specified **level** of aggression:

1. **Level easy**: the boxes are only split around operators and relation symbols that are top level and outside any brackets, fractions or square roots. In our running example, the formula is split into five pieces as illustrated below:

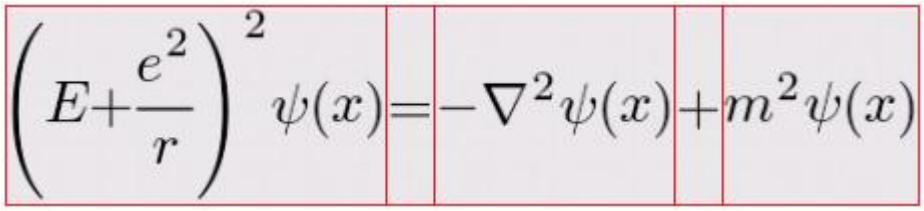


Figure 17: Splitting in level easy

Notice that no splitting occurs around the + operator inside the brackets. This level of aggression will be sufficient as long as `maxWidth` is larger than the width of the widest piece (the leftmost piece in the example). The algorithm takes those five pieces and arranges them into multiple lines using a word-wrap algorithm. So 2 lines if `maxWidth` was spacious enough:

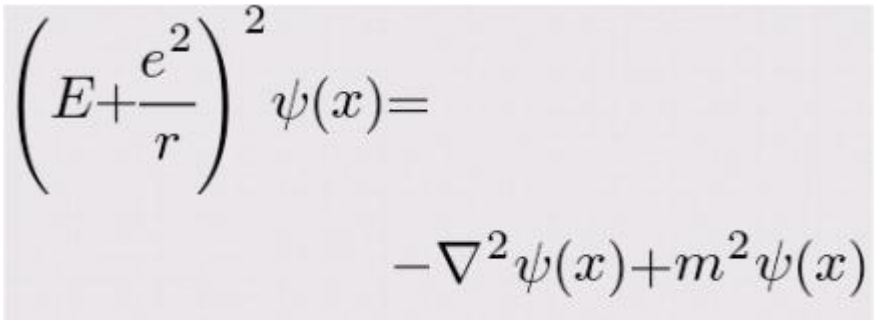


Figure 18: When level easy is sufficient to fit the formula

Or more lines if `maxWidth` was smaller:

$$\left(E+\frac{e^2}{r}\right)^2\psi(x) =-\nabla^2\psi(x) +m^2\psi(x)$$

Figure 19: When `maxValue` is smaller but level easy is still sufficient to fit the formula

2. **Level medium**: is similar to easy but splits inside brackets too, making 7 pieces in total:

$$\left(E+\frac{e^2}{r}\right)^2\psi(x) = -\nabla^2\psi(x) + m^2\psi(x)$$

Figure 20: splitting in level medium

And then applying the word wrap algorithm yields:

$$\left(E+ \frac{e^2}{r}\right)^2\psi(x) =-\nabla^2\psi(x) +m^2\psi(x)$$

Figure 21: formula fitting when level medium is sufficient to fit the formula

The result is less appealing than level easy. But the reward is that the formula can fit inside a smaller width.

3. **Level hard** turns fractions $\frac{e^2}{r}$ into slashes $(e^2)/(r)$, expands square roots $\sqrt{x+1}$ into $\sqrt{(x+1)}$ and then splits inside those too. It turns out that in the example formula, level hard is not an improvement over level medium as there are still 7 pieces, and one of them has become wider:

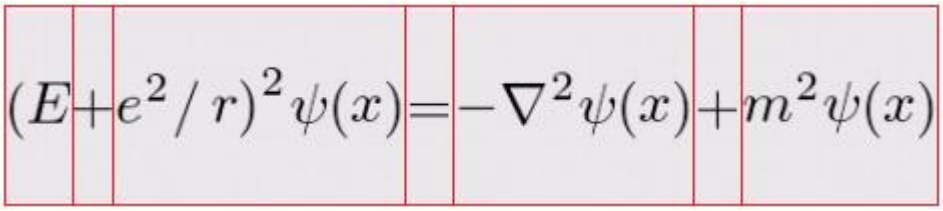


Figure 22: Splitting in level hard

4. **Level nightmare**, splits around every symbol, which for the example formula makes 22 pieces

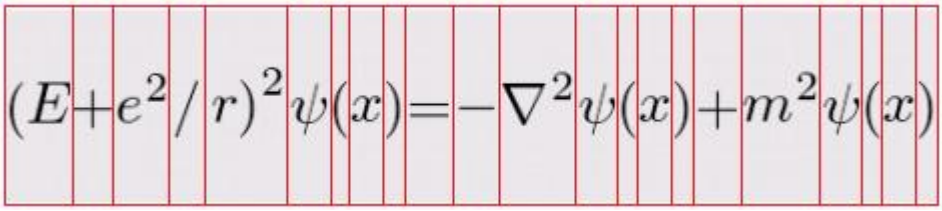


Figure 23: Splitting in level nightmare

Given unrealistically small `maxWidth` values, the result would look like this:

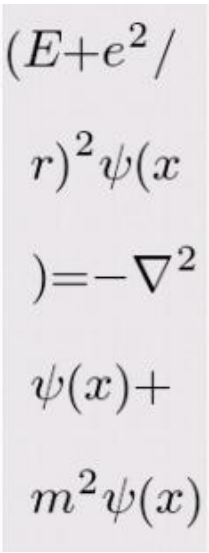


Figure 13: formula fitting under a very small **`maxWidth`**

Figure 14: formula fitting under an unrealistically small **`maxWidth`** (the picture is cropped)

The word-wrap algorithm that arranges the pieces into lines is called Knuth-Plass and is explained in detail in appendix B.

3.3.3 Step 6: Make the font smaller

It is unlikely that the formula fitting algorithm will reach this step, but if all the previous steps fail to fit the formula within `maxWidth`, the last resort is to

reduce the font size until either the formula fits or the minimum specified font size for the boxes is reached. For our running example, see Figure 26 below, the whole formula now fits in less than a centimeter width, as promised.

Figure 26: The example formula after all fitting techniques were exhausted (The picture is cropped)

3.4 Implementing the REDUCible classes

3.4.1 What are the REDUCible classes?

A tree of REDUCible classes is a representation of an algebraic expression, for example:

```
Sequence(Fraction(Sequence(Symbol("1")),Sequence(Symbol("2"))))
```

Represents a fraction of ½, where `Fraction` and `Symbol` are REDUCible classes. The REDUCible tree has the property that the root is a `Sequence` object and every other level will be composed solely of `Sequence` objects in order to support navigation. Section 3.4.4 will explain this in detail.

The difference between a REDUCible tree and a box tree is that the REDUCible tree represents the **meaning** of a formula while a box tree represents just the **visual shape** of a formula.

3.4.2 The function of the REDUCible classes

When the user inputs a formula through the screen keyboard, the user is in reality **building** a REDUCible tree behind the scene. When the user navigates the pointer around the formula using the left and right arrows, the user is in reality **navigating** through the REDUCible tree. Every time the user changes the formula by hitting a key, the REDUCible tree is **translated into a box tree** and this new box tree is drawn on the screen in the place of the old one so

that the changes are reflected to the user. When the user finishes typing the formula and taps enter, the REDUCible tree is **translated to a string in the REDUCE language** and sent to REDUCE for processing (hence the name REDUCible). Therefore, REDUCible trees should support all the following:

1. Easy construction.
2. Navigation.
3. Translation to box trees.
4. Translation to REDUCE.

3.4.3 Constructing a REDUCible tree

I implemented a class called `ExpressionBuilder` which provides methods for constructing a REDUCible tree. Most buttons on the interface keyboard are hooked to methods in an instance of this class. The following code demonstrates how to build $1 + \sqrt{x}$ using the `ExpressionBuilder`:

```
ExpressionBuilder builder = new ExpressionBuilder();

builder.appendSymbol("1");
builder.appendSymbol(S.plus);
builder.appendSqrt();
builder.appendSymbol(S.x);
builder.moveCursorToTheRight(); // explained in the next subsection
```

After this code, `builder` holds internally a REDUCible tree representing $1 + \sqrt{x}$, and storing the position of the cursor at the far right.

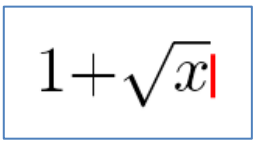


Figure 27: The REDUCible tree constructed by the code above

3.4.4 Navigating through the REDUCible tree

The most important aspect of a REDUCible tree is that it was designed to support left and right **navigation** to random locations in the tree and insertion of items at those random locations. An essential building block of a REDUCible tree is the so-called **`Sequence`**. A `Sequence` is an object that stores a list of REDUCible objects inside it. Each one of these REDUCible objects can either be a leaf, or a node storing a collection of more `Sequence` objects and so forth. Therefore, the REDUCible tree has the property that the root of the tree is a `Sequence` object, and then the levels of the tree

alternate between being composed solely of `Sequence` objects and being composed of REDUCible objects such as `Symbol` and `Sqrt`.

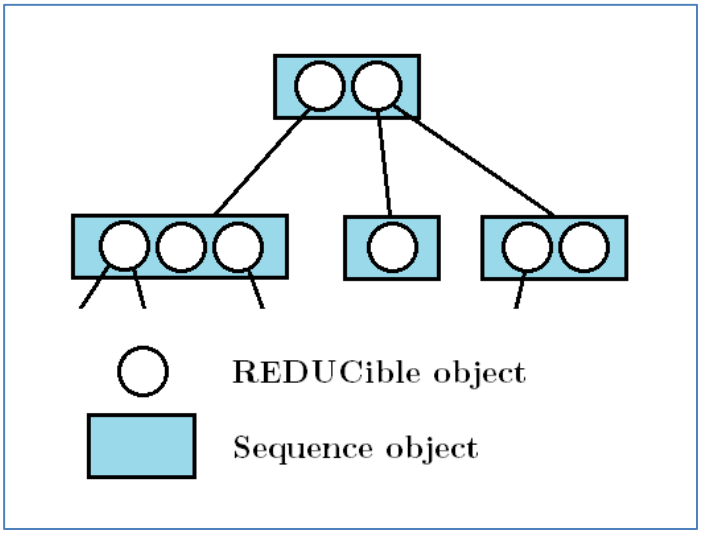


Figure 16: General structure of a REDUCible tree

The following figure is the REDUCible tree representation of $1 + \sqrt{x}$

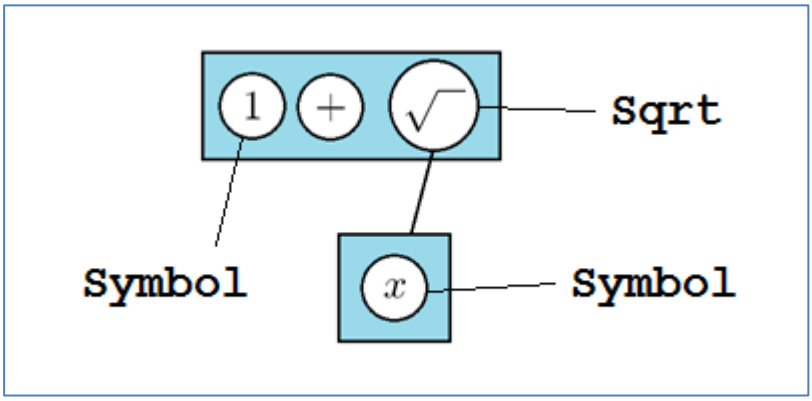


Figure 29: The REDUCible tree representation of $1 + \sqrt{x}$

The reason behind this structure is that it makes navigation easy. In order to support navigation:

1. Every `Sequence` in the tree stores an integer value ranging between 0 and n where n is the number of children in that `Sequence`. This value represents the **cursor position** in the Sequence, so every `Sequence` has one cursor and n+1 cursor positions.

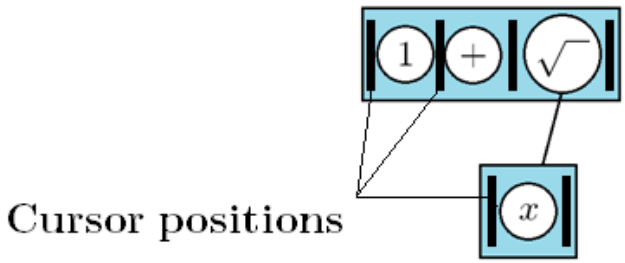


Figure 30: All the cursor positions in the REDUCible tree

2. Only one `Sequence` in the REDUCible tree holds the **focus**. The cursor of that `Sequence` is the only visible cursor and the cursors of all other `Sequences` are hidden.
3. Every `Sequence` object stores a reference to two other `Sequence` objects called **`left`** and **`right`**. `left` and `right` are the `Sequence` objects that take over the navigation when the cursor hits the left end and right end respectively of the `Sequence` object which is referencing them. For example in a `Fraction`[10], the numerator `Sequence` references the denominator `Sequence` as its right `Sequence` so that when the user finishes typing the numerator and clicks the right arrow button, the focus goes to the denominator `Sequence`. Also the opposite is true, the denominator `Sequence` references the numerator as its left `Sequence`. The root Sequence of any REDUCible tree always has a null `left` and a null `right`.

[10] Another REDUCible object

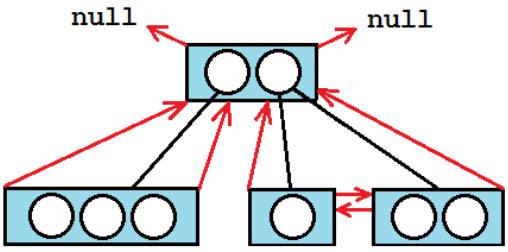
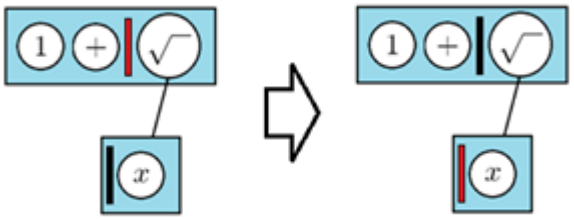


Figure 17: Showing the left and right references in red

The navigation algorithm moves the curser to the left or to the right; the algorithm is symmetrical in both directions. To navigate the cursor one position to the right say, the algorithm asks the `Sequence` that is holding the focus (I will call it **this `Sequence`**) to navigate to the right. This `Sequence` runs one of the following cases:

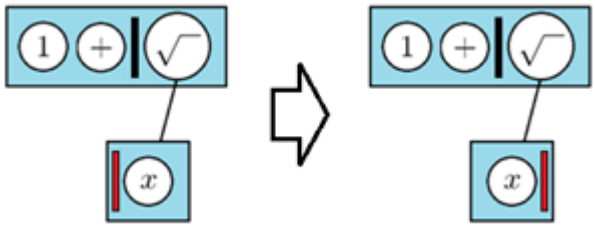
1. If the cursor is not at the rightmost of this `Sequence` and the REDUCible object after the cursor is not a leaf (Contains more `Sequence` objects), then this `Sequence` marks itself as a parent, loses the focus, and gives the focus to the first `Sequence` in that REDUCible object.

   In the figure below, the `Sequence` on the top initially holds the focus, the red bar is the visible cursor. The Sequence at the top marks itself as parent and gives the focus to the `Sequence` at the.



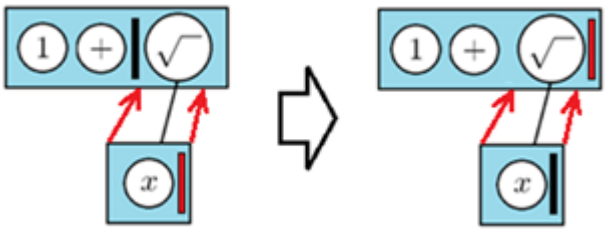
2. If the cursor is not at the rightmost of this `Sequence` and the REDUCible object after the cursor is a leaf, then move the cursor of this `Sequence` one position to the right.

In the figure below, the `Sequence` at the bottom holds the focus and its cursor is followed by a `Symbol` object which is a leaf. Therefore the cursor is simply moved to the right.



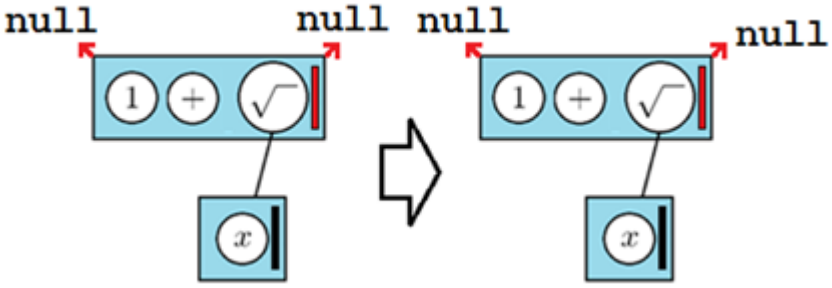
3. If the cursor is at the rightmost of this `Sequence`, and this `Sequence` is not the root of the REDUCible tree (`right != null`) then this `Sequence` loses the focus and the right `Sequence` gains the focus, if the right `Sequence` is marked as parent then also unmark it and move its cursor one position to the right.

In the figure below, the bottom `Sequence` refers to the `Sequence` above as its right `Sequence`. Since the cursor of the `Sequence` below cannot move any further to the right, the focus is handed to the `Sequence` above. Since the `Sequence` above is parent, it moves its cursor one position to the right.



4. If the cursor is at the rightmost of this `Sequence`, and this `Sequence` is the root of the REDUCible tree (`right == null`), then do nothing and the focus remains with this `Sequence`.

In the figure below, the focused cursor cannot move any further to the right, and there is no right `Sequence` to hand the focus to. So nothing happens.



The four cases of the navigation algorithm would appear like this to the user:

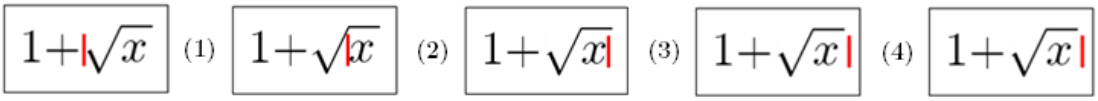


Figure 32: The navigation algorithm at work as the user navigates to the right 4 times

3.4.5 Translating the REDUCible tree to REDUCE

The REDUCible tree is at the end a representation of the user's input. When the user finishes typing the input and taps the enter button, this tree is translated to the input language of REDUCE and sent to REDUCE for processing. For example The REDUCible tree of $1+\sqrt{x}$ is translated to `"1+sqrt(x)"`.

Every REDUCible class implements a recursive method called `toReduce()` which returns the REDUCE representation of that object. Here is the implementation in `Sqrt`:

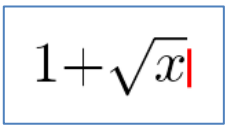
```
public class Sqrt extends WrapperExpression {

	// The body of the square root
	private Sequence body;

	@Override
	protected String toReduce() {
		return String.format("sqrt(%s)", body.toReduce());
	}
```

3.4.6 Translating the REDUCible tree into a box tree

While the user is typing and constructing the REDUCible tree, the user should be able to actually see the thing being typed. This is done by translating the REDUCible tree into a box tree and displaying the box tree to the user every time the user changes the tree. REDUCible classes implement a method `toDrawable()` which does that:

```
public class Sqrt extends WrapperExpression {

	// The body of the square root
	private Sequence body;

	@Override
	public Box toDrawable() {
		return new SqrtBox(body.toDrawable());
	}
```

After constructing the REDUCible tree of $1+\sqrt{x}$, calling `toDrawable()` on the root converts the tree recursively into a box tree that looks to the user like the figure below.



3.5 Implementing the LaTeX parser

After REDUCE processes the input, it returns the result in the form of a LaTeX string. I implemented a parser that can parse the LaTeX into a **tree of boxes**, to make it possible to display the result to the user:

```
String lineOfTeX = "\\frac{x^2}\\mathrm{12}";
TexParser parser = new TexParser(lineOfTeX);
Box boxTree = parser.parse();
```

To implement this parser I received permission from Dr. Arthur Norman to take his C++ implementation of a LaTeX parser (which he wrote in 2004 as part of the CSL LISP system[11]) and translate that into Java. This changed the task of implementing a LaTeX parser into a task of understanding and translating 3K lines of C++ into Java. C++ has some features that were tricky to translate such as pointers to functions and bit fields, there were also problems springing up from the fact that Dr. Norman's parser is designed to generate parse trees that are radically different from my box trees.

3.6 Building the user interface

The interface of my application is just a single Android screen (called **Activity**[12] in the Android terminology). The screen is vertically divided into two areas

1. The math keyboard area which has a fixed height.
2. The input/output display area which occupies the remainder of the screen.

3.6.1 The math keyboard

[11] http://reduce-algebra.svn.sourceforge.net/svnroot/reduce-algebra/trunk/csl/
[12] http://developer.android.com/guide/topics/fundamentals/activities.html

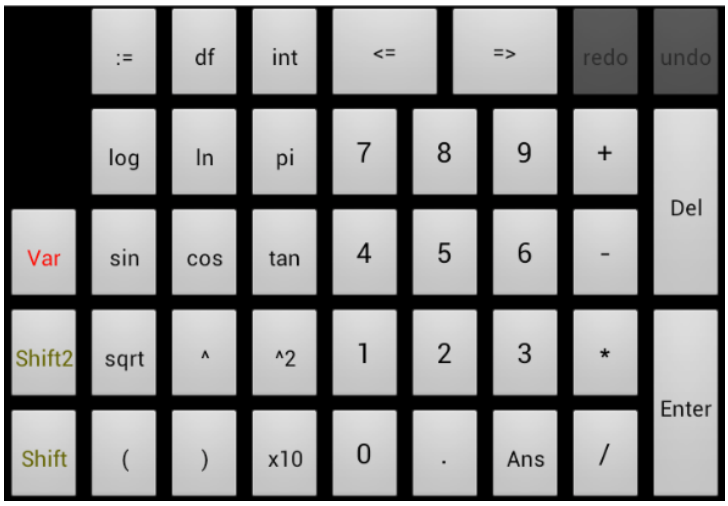
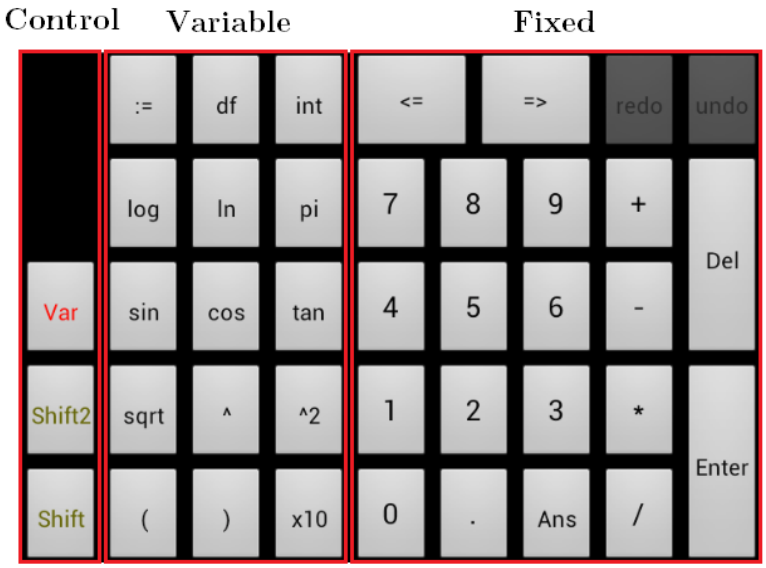


Figure 18: The math keyboard of the app

Given the small area of the phone screen, I can only fit 45 buttons (5 by 9) on the keyboard if the buttons are not to become too small. However the application has over **80 buttons**, and should take into account that the number of buttons may increase in the future. To fit that many buttons, the keyboard is divided into three areas: **fixed**, **variable** and **control** areas.



The fixed area occupies 5 by 5 cells of the keyboard grid and contains 22 of the most commonly used buttons. This area does not change at any time. The variable area on the other hand contains **several** grids of 5 by 3 buttons stacked one on top of the other and only **one grid is visible** at any given time. The buttons in the control area control which grid is visible in the variable area.

For example this is what happens when the button labelled Var is clicked:

Figure 34: Tapping the Var button

This arrangement allows up to 97 buttons + 5 control buttons to fit in the keyboard area.

3.6.2 The input/output display area

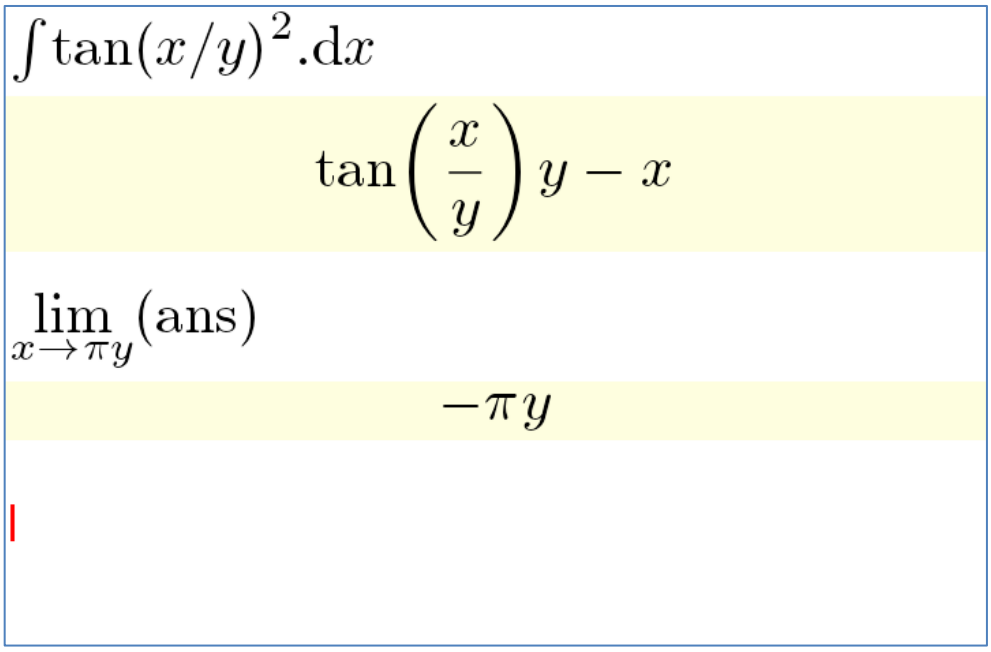


Figure 19: The input/output display area

This area of the screen displays the input and output to the user. It is occupied by an Android `LinearLayout`[13], which is an object that can display a vertical

[13] http://developer.android.com/reference/android/widget/LinearLayout.html

list of things. The `LinearLayout` displays an alternation of `InputView` and `OutputView` which are two custom layout objects that I implemented for the purpose of displaying input and output to the user. Inside every `InputView` and every `OutputView` is an `AndroidGraphics` object displaying a single formula.

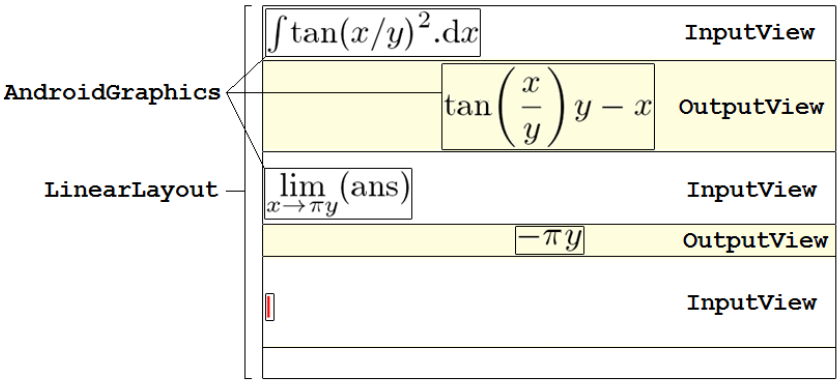


Figure 20: Labeling all the layouts in the input/output display area

See appendix C, for more screenshots of the user interface.

3.7 Summary

This chapter:

- Briefly explained how REDUCE was ported to Android
- Explained the implementation of the math typesetting library (i.e. the box trees).
- Described how the formula fitting algorithm works.
- Explained in detail the concept of the REDUCible trees and how they support construction, navigation and how they are translated into box trees or into the REDUCE language.
- Mentioned briefly how the LaTeX parser was created.
- Described the layout of the user interface.

Chapter 4 Evaluation

In this chapter, I will demonstrate that my work was not only successful in satisfying all the requirements, but that it was high in quality both from a programmer's and from a user's point of view.

4.1 Evaluating the code architecture

4.1.1 Organized code

The source code of this application (more than 11K lines + the modified JLisp) is organized in 7 Java packages and around 70 source files. The DRY principle was applied and every piece of code that is used in several places was moved to a commonly accessible place (typically a base class) and used from there. For example the base class `Box` contains over 500 lines of code that are used by the 15 `Box` subtypes. Also things that are related together are grouped together in one file. For example, there are over 100 LaTeX constants associated with the formula layout rules. These constants determine things like the minimum distance between the numerator and the fraction bar or the default height between the superscript and the base axis lines etc. All these constants are grouped together in one file `C.java`.

```
public static final float minimum_font_size                                    = 14.0000f;
public static final float box_over_box_margin_factor                           = 00.7500f;
public static final float bar_thickness_factor                                 = 00.0400f;

public static final float minimum_display_sqrt_top_margin_factor               = 00.3000f;
public static final float minimum_nondisplay_sqrt_top_margin_factor            = 00.1000f;
public static final float minimum_display_sqrt_bottom_margin_factor            = 00.0000f;
public static final float minimum_nondisplay_sqrt_bottom_margin_factor         = 00.0000f;
public static final float sqrt_bar_nondisplay_thinning_factor                  = 00.8000f;

public static final float script_box_right_margin                              = 00.0500f;
public static final float overunder_script_base_margin_factor                  = 00.1000f;
public static final float base_superscript_margin_factor                       = 00.0500f;
public static final float superscript_axis_display_elevation_factor            = 00.6500f;
public static final float superscript_axis_text_elevation_factor               = 00.5525f;
public static final float superscript_axis_cramped_elevation_factor            = 00.4750f;
public static final float minimum_superscript_elevation_factor                 = 00.1500f;
public static final float maximum_bracket_shortfall_factor                     = 00.1500f;

public static final float normal_display_superscriptAxis_baseAxis_distance_factor  = 00.3300f;
public static final float maximum_display_superscriptFloor_baseAxis_distance_factor = 00.1000f;
public static final float maximum_display_superscriptAxis_baseRoof_distance_factor  = 00.1000f;
public static final float normal_display_subscriptAxis_baseAxis_distance_factor    = 00.3100f;
public static final float maximum_display_subscriptRoof_baseAxis_distance_factor   = 00.1000f;
public static final float maximum_display_subscriptAxis_baseFloor_distance_factor  = 00.1000f;
public static final float minimum_display_subscript_superscript_margin_factor      = 00.1000f;

public static final float normal_text_superscriptAxis_baseAxis_distance_factor     = 00.3725f;
public static final float maximum_text_superscriptFloor_baseAxis_distance_factor   = 00.1000f;
public static final float maximum_text_superscriptAxis_baseRoof_distance_factor    = 00.1000f;
```

Figure 37: All the LaTeX constants live together in one source file

4.1.2 Independence of Android

The `Box` classes do not reference anything that is specific to Android. They do all the drawing through the methods of the interface `Graphics`. The interface implementation in turn calls the GUI API of Android to show the drawing to the user. The `Box` classes perform all the calculations and the drawing completely unaware which platform they are running on and how the drawing actually happens. The behaviour can therefore be easily changed without changing a single line of code in the `Box` classes, only by re-implementing the interface `Graphics`.
To illustrate how this is useful, suppose we want to add a feature to allow the user to email the formula as a GIF image. This would require that the `Box` objects draw themselves to a GIF instead of an Android layout object. All that is required to do this is implement the interface `Graphics` to draw to a GIF instead and inject the new implementation in the `Box` object.

4.1.3 Documentation

Every important method has been commented with Javadoc, in the expectation that another programmer may wish to reuse the code in the future. There are around 1500 lines of Javadoc comments, and these can be exported into an HTML document using the `javadoc.exe` tool.

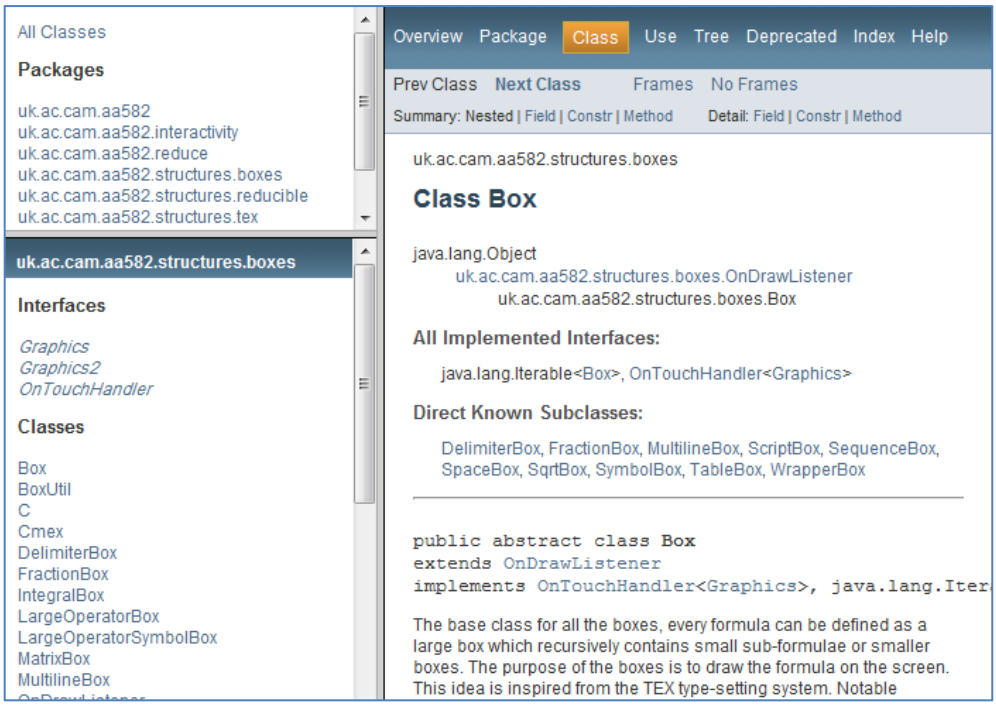


Figure 38: This HTML generated using the Javadoc tool contains all the Javadoc comments that annotate the code

4.2 Evaluating the formula typesetting speed

Box trees are very performance critical. Every time the user hits a key and changes something in the input formula, a new box tree is generated and rendered on the screen quickly so that the user can see the change. According to an influential paper by Robert Miller [**9**] this visual update should take place within 100ms of the click for a seamless user experience. This level of performance was indeed achieved after incorporating two major optimizations.

4.2.1 Multiple layouts vs. a single layout

The first optimization was aimed at lowering memory consumption when drawing the boxes on the screen. Originally the code used to create an Android layout object for every box in the tree. The problem with this approach is that Android layout objects occupy too much memory causing the application used to run out of stack when it attempted to render a tree of boxes only around **10 levels** deep. So the code was changed to create only one layout object for the entire tree. But every box is allocated a sub **area** from that layout object rather than a whole layout object for itself. Changing the code took less than 2 hours thanks to the code organization and maintainability. After this change my evaluation code was able to create and draw random trees of boxes up to **100 levels** deep.

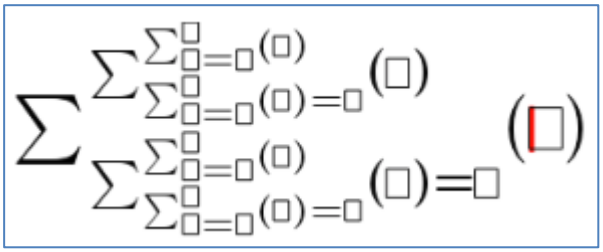


Figure 39: A tree only this deep used to crash the app before it was optimized

4.2.2 Dynamic programming

The second optimization was aimed at speeding up the calculation of the boxes' dimensions before they are drawn. Many of the methods that calculate dimensions such as `width()` take no arguments. These methods recursively traverse the box tree making multiple calls to themselves each level of the tree[14] which makes the procedure exponential in cost, and the drawing very slow.

[14] This is reminiscent of the implementation of the `fib` method, which calls itself twice on every level: `fib(n) = fib(n-1) * fib(n-2),` and hence takes exponential time.

To improve speed, I used the ideas of dynamic programming and traded a lot computational cost for a little memory cost. For every method that takes 0 arguments, I added a field with the same name as the method such that the method is only executed once, and its result is stored in that field. Next time the method is called, it simply returns the stored value in the field without having to recalculate everything. Take for example, the `width()` method in `SequenceBox`:

Original

```
@Override
public float width() {


    float width = 0f;
    for(Box box : this.children) {
        width += box.width();
    }
    return width;
}
```

Optimized

```
protected float width =
    Float.NEGATIVE_INFINITY;

@Override
public float width() {
    if(optimize && width >= 0)
        return width;

    width = 0f;
    for(Box box : this.children) {
        width += box.width();
    }
    return width;
}
```

There is a method `forget()` which is used to clear all the optimization fields when the box width changes. However, box trees are meant to be mostly immutable and so calls to `forget()` happen only once or twice in the lifetime of a box tree. I optimized 50 methods in the project, and added a global Boolean field `optimize` to switch between optimized and not optimized.

Formula	Time to draw	
	Not optimized	Optimized
22	<1ms	<1ms
$y^2\sqrt{x}$	6ms	2ms
$\frac{\log(\tan(x)^2+1}{2}$	700ms	6ms
$z = \frac{\log(\tan(x)^2+1)}{2} - y^2 + \sqrt{x}$	3,000ms	10ms
$\left\{u = \frac{\log(\tan(x)^2+1)}{2}, v = y^2 + \sqrt{x}\right\}$	29,000ms	12ms

The measurements above were performed by evaluation code on a Galaxy Nexus handset, there is up to 15ms variance in the numbers, but the differences

are so large that this inaccuracy is irrelevant. It is evident now that the drawing time after the optimization is **well below** the 100ms benchmark.

4.3 Beauty of the formula typesetting

Beauty is in the eye of the beholder, but this section attempts to assess the quality of the formulae drawn by the math typesetting library.

4.3.1 Comparison with LaTeX

There is a general consensus that the formulae rendered by LaTeX are very beautiful and close to perfection. If one types "most beautiful math typesetting" into the Google search engine, one will discover that half of the top 20 results are related to LaTeX. Therefore, I tried as much as possible to approximate LaTeX in my math typesetting library. This gives us a straightforward way of evaluating the "beauty" of the math typesetting simply by comparing it side by side with that of LaTeX:

Math typesetting library	**LaTeX**
$\tan\left(\frac{x}{y}\right)^2 + 1$	$\tan\left(\frac{x}{y}\right)^2 + 1$
$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$	$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
$x = \int_a^b v.\mathrm{d}t$	$x = \int_a^b v.\mathrm{d}t$
$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$	$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
$\gamma = \frac{1}{\sqrt{1 - (\frac{v}{c})^2}}$	$\gamma = \frac{1}{\sqrt{1 - \left(\frac{v}{c}\right)^2}}$

Without the labels, it is difficult to tell which formulae were made by LaTeX.

4.4 Evaluating the algebra capabilities

4.4.1 REDUCE capabilities

The algebra features of this application are provided by the REDUCE algebra system[15]. This system has grown over more than 40 years and became very sophisticated to such an extent that it is difficult to list all the functionality that it can provide. All this functionality is at the disposable of our application, but only part of it was exposed to the user.

4.4.2 Exposed features

In order not to overwhelm the developer or the user, only a basic subset of the REDUCE features was exposed. It includes:

- Simple arithmetic (+ - * /)
- Simplification of expressions
- Expression factorization
- Numerous functions (trig, hyperbolic, log, factorial etc...)
- Complex numbers
- Differentiation and integration.
- Big sum and big product
- Limits
- Variable initialization (m := 2)
- Manipulation of polynomial equations
- Solving simultaneous and polynomial equations
- Matrices

Adding more features to this list is easy as they are already available from REDUCE, in most cases it is only a matter of adding another button, adding another REDUCible class and connecting the two in an event handler. The list of features that I have exposed already surpasses the features of all algebra apps available for Android except for one, as described later in Section 4.8

4.5 Evaluating the usability

It was mentioned in the project proposal that HCI testing will not be feasible. So evaluating the app usability will be done by listing some of the features that were not part of the specifications but were implemented to improve usability.

[15] http://reduce-algebra.com/

4.5.1 Wait animation

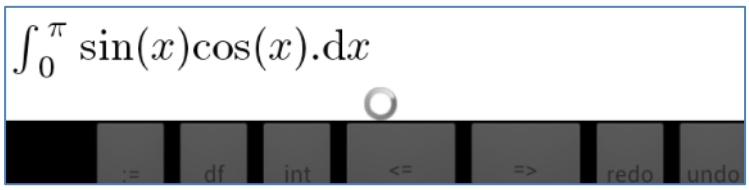


Figure 24: A small animated circle appears when REDUCE is processing user input

REDUCE can take up to a few seconds to process certain types of inputs (such as integration with limits). During that time, a wait animation is displayed to reassure the user that the input is being processed.

4.5.2 Undo and redo

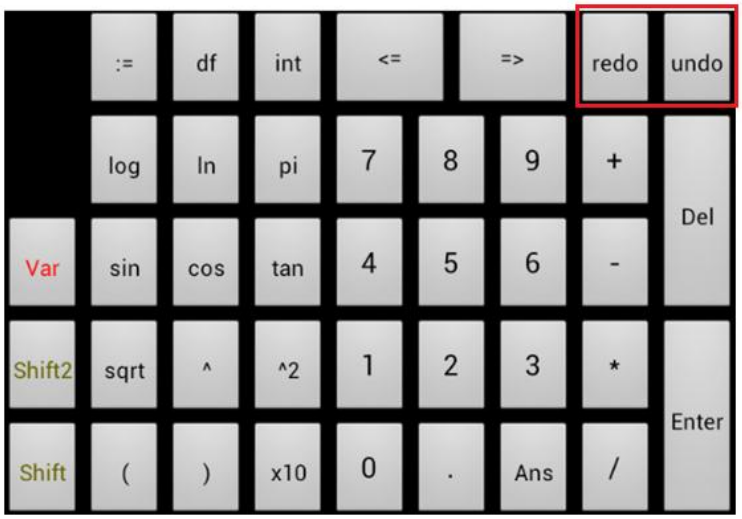


Figure 25: The application keyboard, with the undo and redo buttons highlighted

The undo button gives the opportunity for the user to backtrack when something unexpected happens. The application maintains an **undo stack** where all the actions of the user are recorded, each action alongside its inverse. When the user taps undo, the last action is popped from the undo stack, its inverse is executed and then the action is put in the **redo stack**.

4.5.3 Touch to navigate feature

To navigate the cursor around the formula, the normal way is to use the left and right buttons in the keyboard. I also implemented a shortcut for navigation which is to touch the formula where you want the cursor to go. I implemented this feature unaware how many users expect it naturally to be available. Of all the people who used this app for the first time and, while typing a formula,

reached a point where they needed to move the cursor to a certain position, most of them touched that position with confidence that this will work, and continued typing their input.

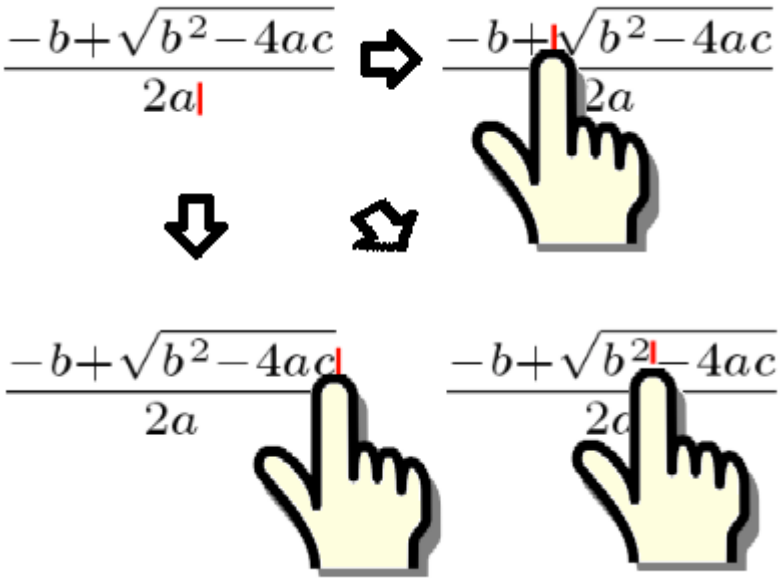


Figure 26: Touching a formula navigates the cursor to the touched location

4.5.4 Touch to copy feature

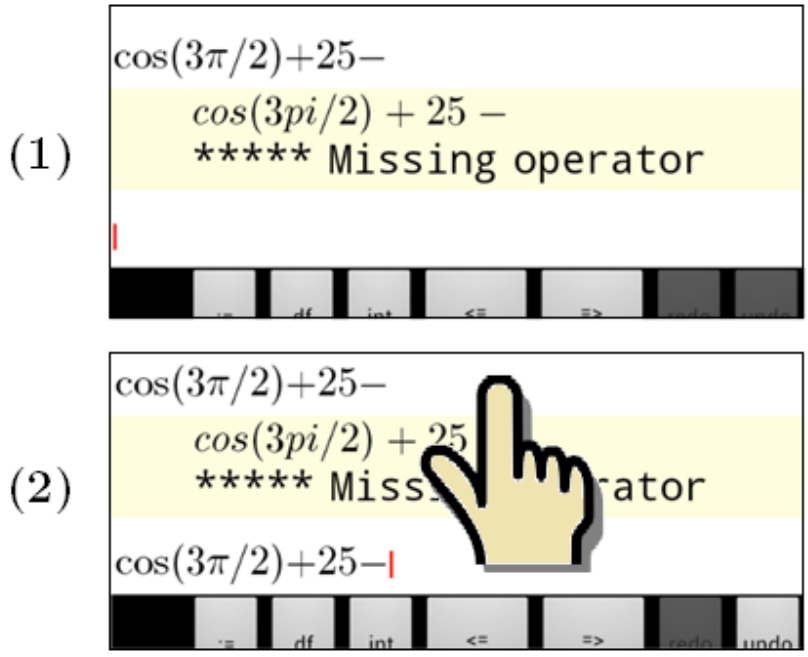


Figure 27: (1) the user mistyped the formula (2) but it can be fixed without having to retype it

Another useful feature allows the user to touch a previous formula to copy it down to the next input area. This is useful when the user entered the first formula incorrectly because it allows the user to copy the faulty formula and fix it rather than having to retype it again.

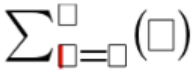
4.5.5 Consistency between REDUCE and the user

The user and REDUCE are interacting with each other through the interface of this application, this interface is therefore responsible for maintaining a consistent view between the two which is not trivial. For example when the user inputs "$x\cos(x)$", the user really means "x multiplied by $\cos(x)$". It would be a surprise to the user if REDUCE responded: "would you like to declare xcos as a new function?". To avoid such surprises, the translation from the REDUCible tree to the REDUCE language (See subsection 3.4.5) was made more intelligent to know where to explicitly insert a multiplication sign so that REDUCE is consistent with the user. Another example is when the user inputs "$x1 + 3$", and finds that the answer is "$x_1 + 3$" because REDUCE treats every number following a variable name as a subscript of that name. So again to maintain a consistent view, the translation from the REDUCible tree to the box tree (see subsection 3.4.6) was made more intelligent to display "$x1 + 3$" correctly as "$x_1 + 3$".

4.5.6 Automatic closing of open brackets

REDUCE does not accept input with unclosed brackets, such as `"sin(x"`. So the translation to REDUCE closes any open brackets. For example if the user enters $\frac{\sin(x}{\cos(x)}$, the translation inserts the missing bracket in the right place like this: `"sin(x)/cos(x)"`.

4.5.7 Empty rectangles for empty sequences



An empty `Sequence` object is translated to a `SpaceBox` with the `borderVisible` field set to true. This appears to the user as a little empty rectangle which acts as a reminder that something should go in this place.

4.6 Room for improvement

4.6.1 Keyboard labels

The buttons of the keyboard are currently labelled in text. This can be improved by labelling them with icons showing real mathematics instead (e.g. π instead of pi). However preparing hundreds of icons and changing the buttons is very time consuming even though not technically difficult.

4.6.2 Remembering the activity state

When the user navigates from one screen (or activity in Android terminology) to another, all the screens that the user visits are stored in memory so that when the user returns back to a previous screen, it is found intact. But occasionally Android decides to kill one of the activities in memory, in which case when the user returns to that screen, the screen is restarted. Android mandates that in this case any activity should be able to recall the state at which the user left it last time and reconstruct itself exactly the way it was. This is currently not implemented for the main screen of this project because the underlying state is very complicated, the state must remember the internal workings of JLisp and REDUCE themselves, and this problem could not be addressed within the project time frame. At the moment, if Android kills and then restarts the application it will simply start from scratch and previous input is not preserved.

4.7 Product stability and known issues

4.7.1 Testing

Most of the code I wrote for the application (such as the math typesetting library) is GUI centric, and is therefore difficult to unit test automatically. But the fact that the application runs on my Galaxy Nexus gave me the idea of testing it in real life by using it in my daily work as mentioned in section 2.4 . I was able to discover and fix a large number of nontrivial bugs.

The final version of the application was verified to work on the following Android devices, by trying a number of algebra calculations.

1. Galaxy Nexus (Android 4.0)
2. Galaxy Tab (Android 2.3)
3. HTC Desire (Android 2.3)

4.7.2 The stack size concern

There are 2 threads running in this application:

1. The main thread that manages the UI.
2. The REDUCE thread which is running REDUCE in the background.

REDUCE in particular consumes plenty of stack. The reason is that tail recursion is very common in the sources of REDUCE, in the hope that the LISP system running REDUCE optimizes tail recursion. JLisp does not do it because tail calls are impossible in Java at the JVM level. The amount of stack that is consumed by REDUCE is tolerable on the PC but not on Android. Every thread on Android is only allocated 8KB of stack which is not enough for REDUCE. For example, sending $\int_1^2 x.dx$ for input caused the application to crash. To solve this problem, Android allows me to change the stack size of spawned threads using a constructor parameter `stackSize`.

public **Thread** (ThreadGroup group, Runnable runnable, String threadName, long stackSize)

I used this parameter to specify a stack size of 20MB. And it solved the problem on all Android devices that were used for testing this. The documentation of `stackSize` however, remains worrying:

stackSize a stack size for the new `Thread`. This has a highly platform-dependent interpretation. It may even be ignored completely.

4.8 Comparison with existing software

In this section, I compare our application with the best algebra applications available on Google Play.

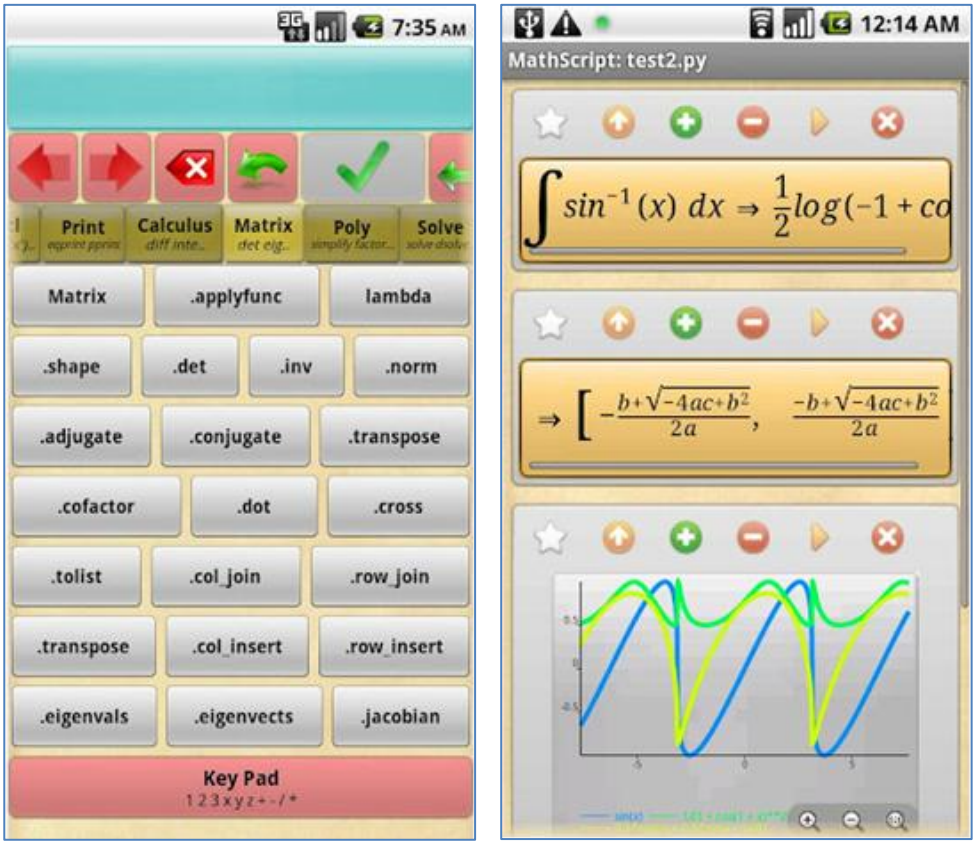
4.8.1 WolframAlpha

WolframAlpha comes from the same developers of Mathematica. This android app is marketed as a knowledge engine. It answers questions not only in

algebra (which is the main focus here), but also in an extravagantly large number of topics. Over 250 of these topics are listed in the app's description page[16]! The algebra capabilities of WolframAlpha are without doubt much more than what our application exposes. And even comparable to what is delivered by REDUCE itself. However, my application surpasses WolframAlpha in 2 ways:

1. The WolframAlpha app is only a client, and the actual knowledge engine is a web service. So an Internet connection is required to use it. My application on the other hand runs entirely on the Android device.
2. WolframAlpha shows the output, but not the input, in true math notation. While my application shows both the output and the input in true math notation.

4.8.2 MathScript



MathScript is described by its creators as:

> "The first application to bring comprehensive mathematics to Android, and to our knowledge, it is the only such application available"

[16] http://play.google.com/store/apps/details?id=com.wolfram.android.alpha

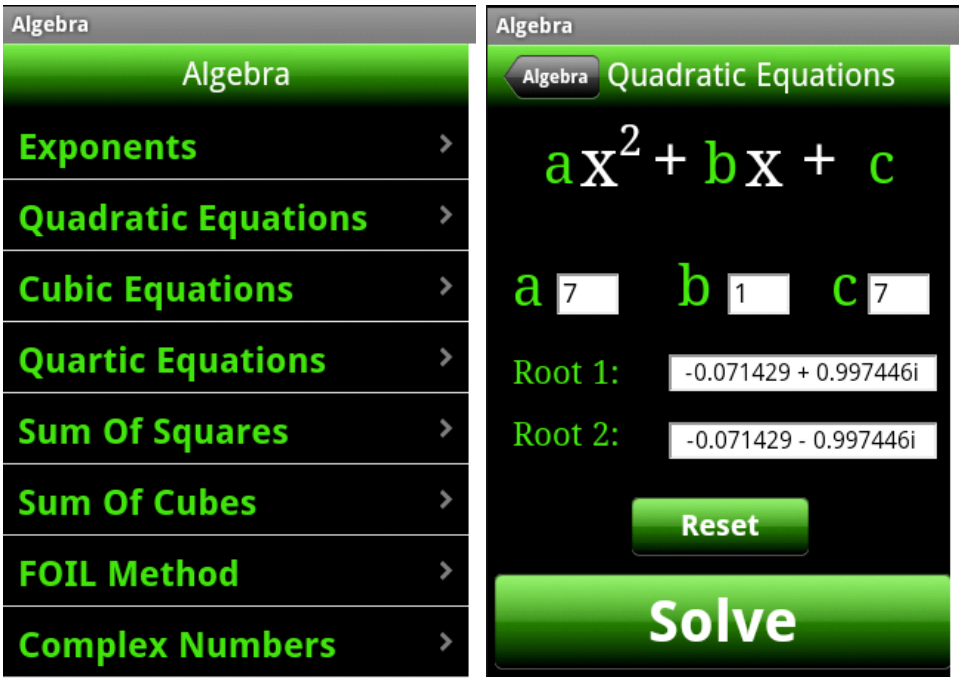
And this description is correct; no other application on Google Play offers algebra capabilities similar to what MathScript can do, with the exception of WolframAlpha. But it was mentioned that WolframAlpha is a web service and not running on Android. To compare MathScript with my application:

1. MathScript's algebra capabilities are listed on the app's website [**10**] and they surpass what our application currently **exposes**.
2. MathScript's algebra capabilities are less than what is available from REDUCE. With the exception of the graph plotting feature, which REDUCE does not have since graph plotting is not generally considered an algebra feature even though it is nice to have.
3. The interface of MathScript is more colourful than the interface of my application. Clearly the creators spent more effort on graphics design.
4. MathScript uses natural math notation for both the input and the output as does my application.

4.8.3 Other applications

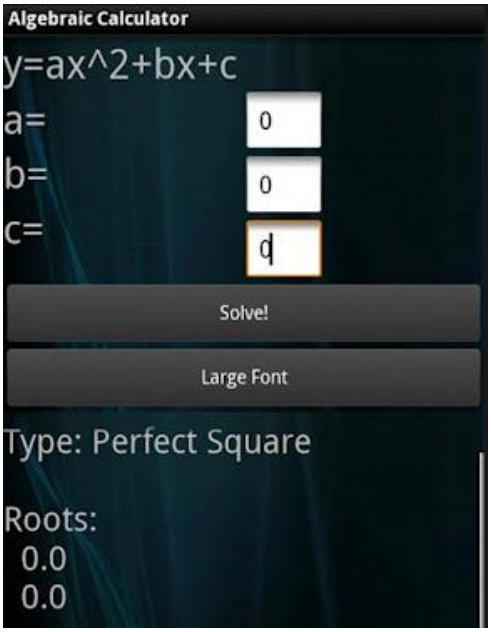
All other app available on Google Play can only perform simple algebra, less than what is exposed by my application, and much less of course than what is available from REDUCE. This subsection mentions 3 applications:

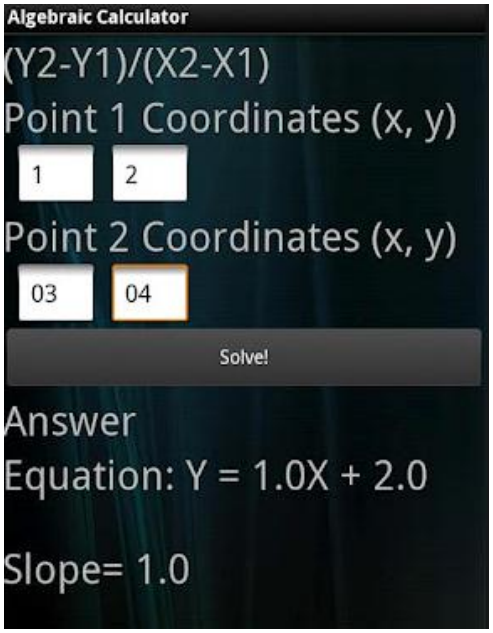
1. **Algebra Calculator Pro**



This application is available for free, and it can do a small range of Algebra. But it takes input in text fields.

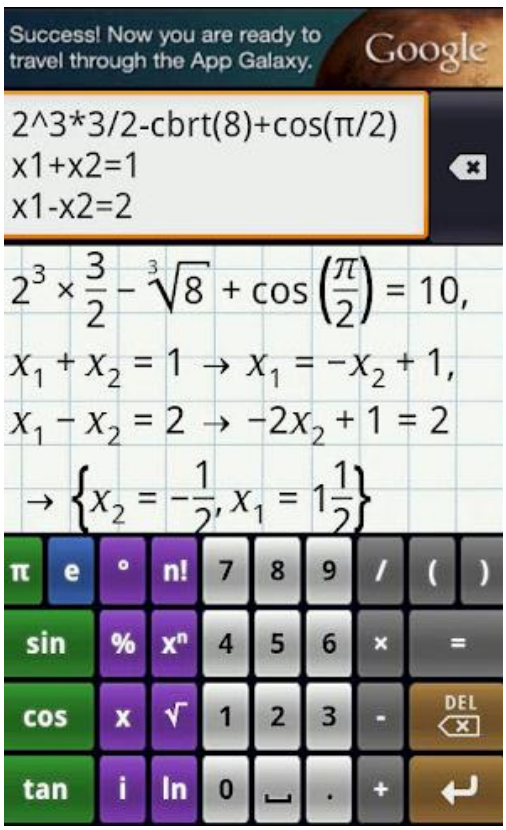
2. Algebraic Calculator

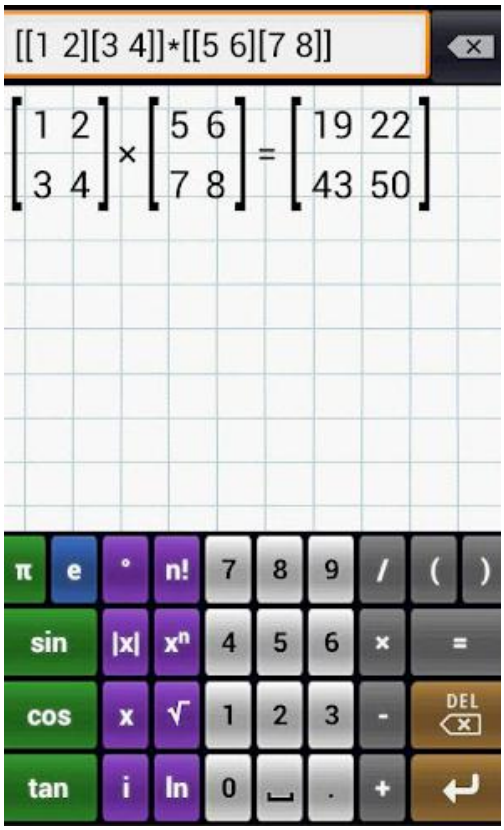




This application is very similar to Algebra Calculator Pro.

3. Mathlab Graphing calculator





This app shows output (but not input) in true math notation. It is more specialized in graph plotting and does not have extensive algebra.

4.9 Summary

This chapter:

- Evaluated the code architecture and maintainability.
- Measured the responsiveness of the application.
- Measured the elegance of the math typesetting by comparing it with that of LaTeX.
- Enumerated the algebra features that were exposed through the app interface, and hinted to how much more is provided from REDUCE.
- Mentioned some of the UI features that were added to improve the user experience.
- Mentioned some of the problems that time did not permit to solve.
- And finally compared our application with other algebra software available from Google Play.

Chapter 5 Conclusions

In conclusion, the project was well planned, the implementation process was streamlined and all the requirements were satisfied with a high quality product. I extended my knowledge into many new areas and I quite enjoyed writing the code, especially for the math typesetting library (As described in section 3.2). The fact that this application offered something new to the Android platform was a very important motivation.

5.1 Additional work

In the future, the following work would need to be added to improve the application:

1. Expose more of the features of REDUCE such as programming and Taylor series expansions.

2. Improved the graphic design of the interface to make it more appealing, and label the keyboard buttons with true math symbols instead of text.

3. Add a preferences activity to allow the user to specify the font size and theme etc.

4. Provide our application with an **intent filter**. This will make it available to other applications on the phone that need its powerful algebra capabilities.

5.2 Final remark

The application created for this project uses the open source JLisp and REDUCE, and therefore must remain open source. The application is available for free on Google Play under the name "Android REDUCE", and the source code will be published on SourceForge.com for other developers to build over the foundation of my work.

References

[1] Anthony C. Hearn. (2004, February) REDUCE Computer Algebra System. [Online]. www.reduce-algebra.com/docs/reduce.pdf

[2] Arthur C. Norman, "Further Evaluation of Java for symbolic computation," in *ACM symposium on Symbolic and Algebraic Computation*, 2000, pp. 258-265.

[3] Donald E. Knuth, *The TeXbook.*: Addison-Wesley, 1984.

[4] Michael Downes, "Breaking equations," in *TUGboat*, Providence, 1997.

[5] Google. The Developer's Guide. [Online]. developer.android.com/guide/

[6] John Sonmez. (2011, April) Introduction to Android Development. Video. [Online]. www.pluralsight-training.net/microsoft/Courses

[7] Martin Fowler and Others. (2001, February) Manifesto for Agile Software Development. [Online]. agilemanifesto.org

[8] Ken Schwaber, *Agile software development with Scrum.* Upper Saddle River, NJ: Prentice Hall, 2002.

[9] Robert B. Miller, "Response time in man-computer conversational transactions," in *AFIPS Fall Joint Computer Conference Vol. 33*, 1968, pp. 267-277.

[10] The Think Tanks. MathScript: The complete math solution for Android Mobiles. [Online]. http://www.touchthinktanks.com

Appendix A The `Box` subtypes

In this appendix I list all the `Box` subtypes. And describe each one of them briefly.

A.1 SymbolBox

This box simply displays a string of characters. The height of the box is the bounding height of that string, and the width of the box is the advancing width of the string.

Children: None.

Style of children: Does not have children.

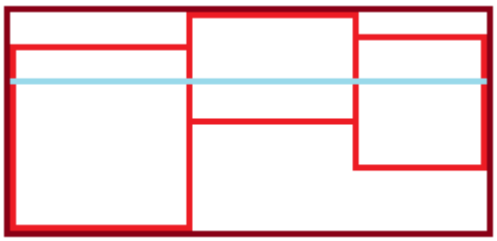
A.2 SpaceBox

This box is just an empty space with a specified width and height. The axis height is exactly one half the height of the box.

Children: None.

Style of children: Does not have children.

A.3 SequenceBox

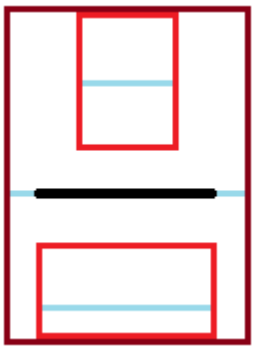


This box arranges its children in a sequence such that their axes align. The axis line of this box is the same as the aligned axis lines of its children.

Children: 0 or more children.

Style of children: The style of the children is the same as the style of the parent.

A.4 FractionBox



This box arranges its 2 children, numerator and denominator, vertically and draws a horizontal line between them. The distance between the numerator's axis line and the fraction bar is a default distance unless the depth of the numerator is too big that margin between it and the fraction bar becomes less than a certain minimum in which case the numerator is positioned according to that minimum. Same applies for the denominator. The box adds a small margin to the left and to the right of the fraction. And the axis line of the `FractionBox` is aligned with the fraction bar.

Children: 2 children, numerator and denominator.

Style of children:

`FractionBox`	Numerator	Denominator
Display	Text	Cramped text
Cramped display	Cramped text	Cramped text
Text	Script	Cramped script
Cramped text	Cramped script	Cramped script
Script	Script script	Cramped script script
Cramped script	Cramped script script	Cramped script script
Script script	Script script	Cramped script script
Cramped script script	Cramped script script	Cramped script script

A.5 SqrtBox

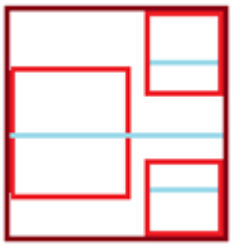
This box surrounds the body with a square root sign. There are several square root signs of various sizes, and this box picks the smallest sign that can surround the body. If the body is too large, the sign can be made arbitrarily large by building a tower of vertical segments. The axis line of the SqrtBox is aligned with the axis line of the body inside it.

Children: One child, called body.

Style of children:

`SqrtBox`	body
Display	Cramped display
Cramped display	Cramped display
Text	Cramped text
Cramped text	Cramped text
Script	Cramped script
Cramped script	Cramped script
Script script	Cramped script script
Cramped script script	Cramped script script

A.6 ScriptBox



This single box can position both the superscript and the subscript of a single base such that they are not too high, too low or overlapping with each other. The superscript is placed at the minimum elevation that satisfies the following:

1. The axis line of superscript should be at least X_1 above the axis line of base.
2. The axis line of superscript should be at most Y_1 below the roof of base.
3. The superscript floor should be at most Z_1 below the axis line of base.
4. The superscript and subscript should be at least a margin M apart.

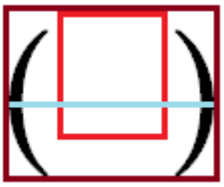
The constants X_1, Y_1, Z_1 and M are suitably chosen to make the result resemble that of LaTeX. The subscript has similar rules (In the other direction) and another set of constants X_2, Y_2 and Z_2. The axis line of the `ScriptBox` is the same as that of its base.

Children: 3 children: base, superscript, and subscript. Either superscript or subscript may be omitted bringing the number of children down to 2.

Style of children:

`ScriptBox`	**Base**	**Superscript**	**Subscript**
Display	Display	Script	Cramped script
Cramped display	Cramped display	Cramped script	Cramped script
Text	Text	Script	Cramped script
Cramped text	Cramped text	Cramped script	Cramped script
Script	Script	Script script	Cramped script script
Cramped script	Cramped script	Cramped script script	Cramped script script
Script script	Script script	Script script	Cramped script script
Cramped script script	Cramped script script	Cramped script script	Cramped script script

A.7 DelimiterBox

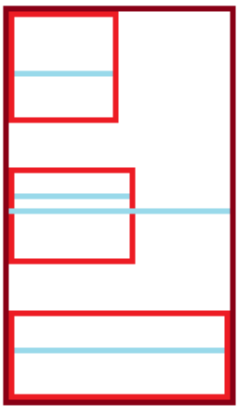


Surrounds its child with delimiters (parenthesis, square brackets, curly brackets or absolute value bars). The delimiters are positioned such that the axis line of the child goes through the centre of the delimiters. The size of the delimiters is chosen such that the delimiters completely surround the child except for a small shortfall distance (example in the figure above). The delimiters can be made arbitrarily large for this purpose. The axis line of the `DelimiterBox` is aligned with that of its child.

Children: One child, named body.

Style of children: The style of the child "body" is the same as the style of the parent `DelimiterBox`.

A.8 MultilineBox

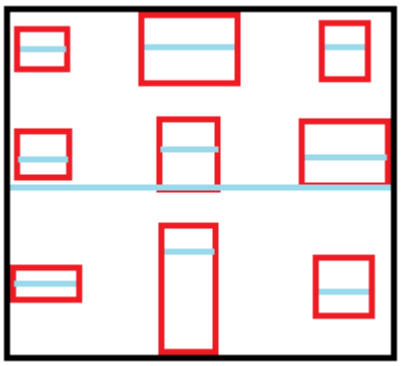


Arranges all the children one on top of the other, with a specified margin between them, and an option to align them to the left, to the right or centre them. The width of the `MultilineBox` is the width of its widest child. And the axis height is exactly half the height.

Children: 0 or more children.

Style of children: Same style as their parent.

A.9 TableBox



This box arranges its children in a 2 dimensional grid, such that the width of each column is the width of the widest child in that column, and the height of each row is the height of the tallest child in that row. There is a fixed margin between every two adjacent rows and every two adjacent columns. The axis height of the `TableBox` is exactly half its height. Wrapping the `TableBox` inside a `DelimiterBox` should look like a matrix.

Children: 0 or more children.

Style of children: Same style as their parent.

Appendix B Knuth-Plass

I explained in section 3.3 that the fitting algorithm uses a **word wrap** algorithm to arrange the pieces into lines. The simplest word wrap algorithms puts as many pieces as possible in the first line and then as many pieces as possible in the second line etc... This minimizes the sum of the empty spaces in each line. Algorithms like this are cheap and simple to implement but do not look very good. For example, it can fit `"uuu xx yy zzzzz"` in a line width of 6 as follows (assuming that each sequence of letters is a piece):

```
|uuu xx| (0 empty spaces)
|yy    | (4 empty spaces)
|zzzzz | (1 empty space)
```

The sum of the empty spaces is 0+4+1 = 5 which is the minimum possible value. On the other hand I implemented a better algorithm called **Knuth-Plass** which is the algorithm used in LaTeX. This algorithm minimizes the **sum of the squares** of the empty spaces in each line producing a more pleasing result. For example applying knuth-plass on the previous example:

```
|uuu   | (3 empty spaces)
|xx yy | (1 empty space)
|zzzzz | (1 empty space)
```

The sum of the squares is 9+1+1 = 11, which is the minimum possible value. The simple implementation of Knuth-Plass has a cost exponential in the number of pieces, but it can be implemented in $O(n^2)$ time and space using dynamic programming.

Knuth-Plass takes 2 arguments

1. An array of boxes or `pieces`, each piece has a known width.
2. `maxWidth`, which is the maximum width of the lines.

Assuming of course that the width of the widest piece is < `maxWidth`, and that the sum of the widths of the pieces is > `maxWidth`. The algorithm returns a `MultlineBox` object, with each line in that object containing a group of those pieces stacked side by side in a `SequenceBox`.

To solve the problem in $O(n^2)$, we define a cost function $c(i,j)$, which computes the cost of a line containing all the pieces from i to j :

$$c(i,j) = \left(maxWidth - \sum_{k=i}^{j} pieces[k].width() \right)^2$$

There is however a special case to this. If the value inside the brackets is negative (which means that the pieces from i to j cannot fit in `maxWidth`) then the cost $c(i,j) = \infty$. The cost of the optimal solution $f(piece\ count)$ can be defined recursively:

$$f(n) = \begin{cases} c(1,n) & if\ c(1,n) < \infty \\ \min\limits_{1 \le m < n} \left(f(m) + c(m+1,n) \right) & if\ c(1,n) = \infty \end{cases}$$

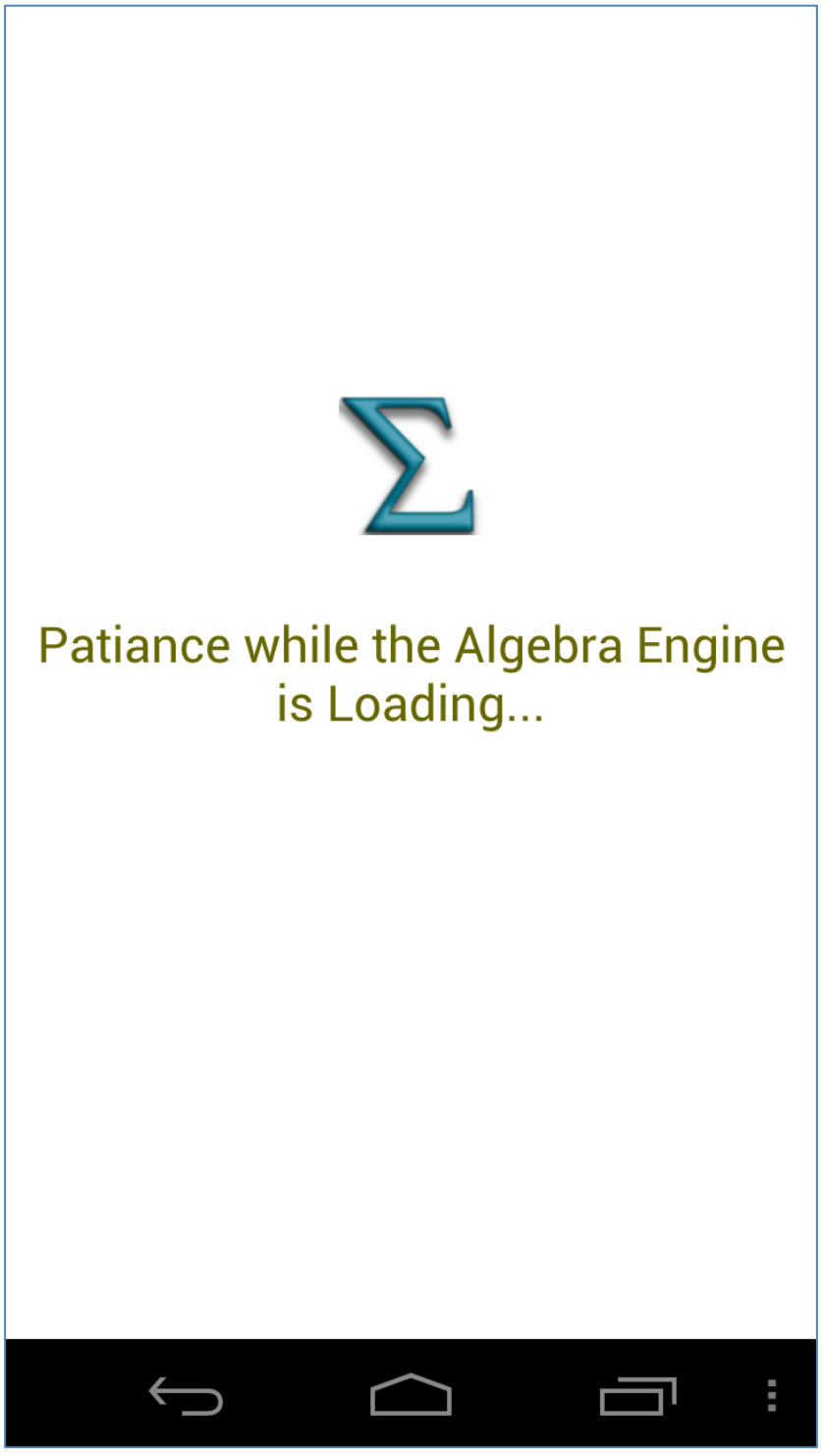
This recursive function can be implemented in $O(p^2)$ time and space where p is the total count of the pieces. This is done by creating 2-dimensional arrays: `sum`, `cost` and `minCost` such that `sum[i,j]` is the sum of the widths from piece i to piece j, and so on.

Here I present the complete code of my implementation:

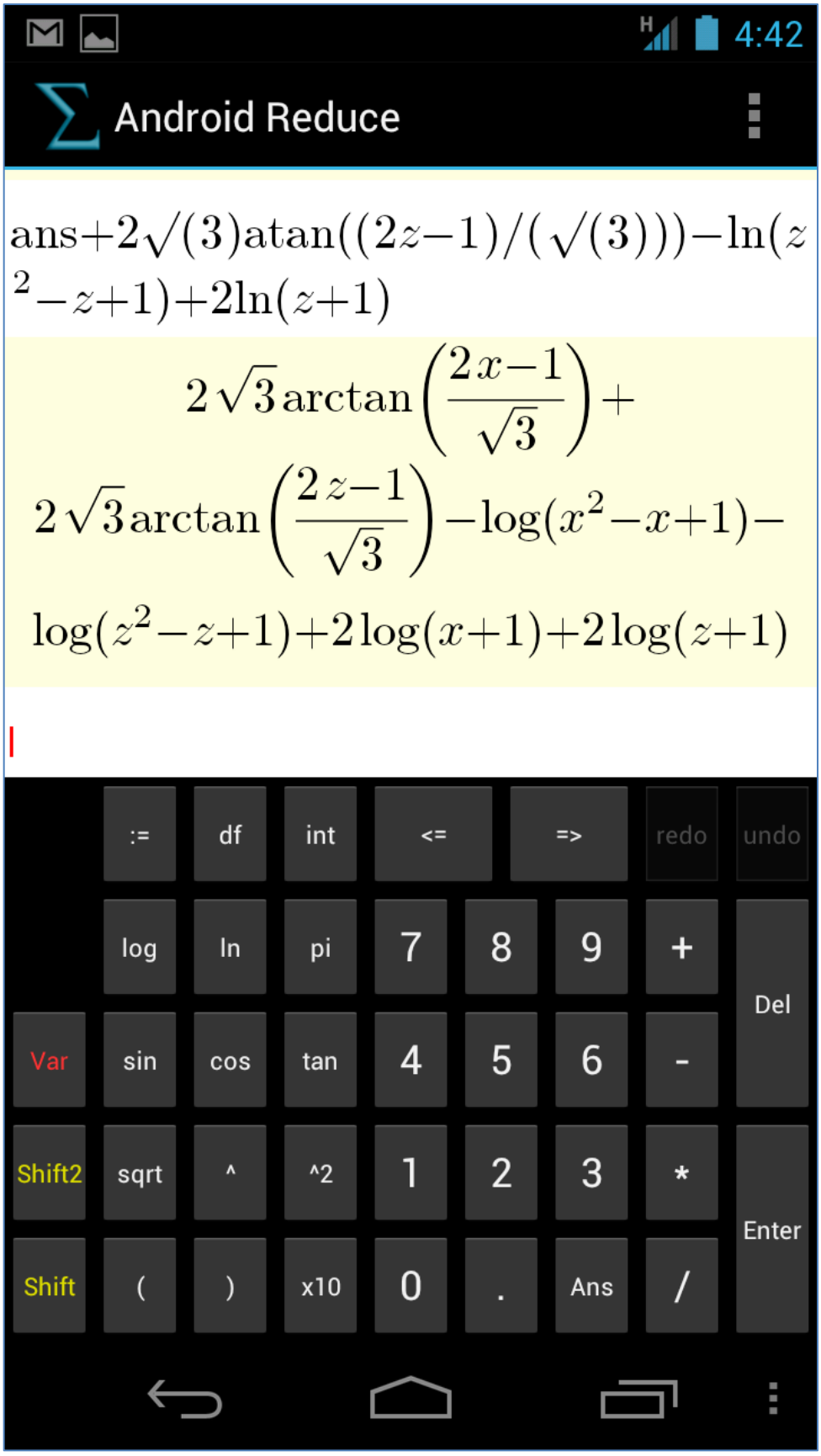
```
private static Box knuthPlass(Box[] pieces, float maxWidth) {

	int pieceCount = pieces.length;
	if(pieceCount == 1) return pieces[0];

	// calculate 'sum' in O(n^2)
	float[][] sum = new float[pieceCount][pieceCount];
	for(int i=0; i<pieceCount; i++) {
		sum[i][i] = pieces[i].width();
		for(int j=i+1; j<pieceCount; j++) {
			sum[i][j] = sum[i][j-1] + pieces[j].width();
		}
	}

	// calculate 'cost' in O(n^2) using 'sum'
	float[][] cost = new float[pieceCount][pieceCount];
	for(int i=0; i<pieceCount; i++) {
		for(int j=i; j<pieceCount; j++) {
			float diff = maxWidth - sum[i][j];
			if(diff < 0) // if words don't fit on the line
				cost[i][j] = Float.POSITIVE_INFINITY;
			// else cost is the square of the difference between
			// the width of the pieces and the width of the line
			else cost[i][j] = diff * diff;
		}
	}
```

```
        // calculate 'minCost' and 'pointer' in O(n^2) using 'cost'
        float[] minCost = new float[pieceCount];
        int[] pointer = new int[pieceCount];
        minCost[0] = cost[0][0];
        pointer[0] = 0;
        for(int j=1; j<minCost.length; j++) {
              if(cost[0][j] < Float.POSITIVE_INFINITY) {
                    minCost[j] = cost[0][j];
                    pointer[j] = 0;
              } else {
                    float minJcost = Float.POSITIVE_INFINITY;
                    int minK = -1;
                    for(int k=j-1; k>=0; k--) {
                          float kCost = minCost[k] + cost[k+1][j];
                          if(kCost < minJcost) {
                                minJcost = kCost;
                                minK = k;
                          }
                    }
                    minCost[j] = minJcost;
                    pointer[j] = minK+1;
              }
        }

        // extract the lines using 'pointer' and arrange each group
        // pieces representing a line inside a SequenceBox
        LinkedList<Box> lines = new LinkedList<Box>();
        int index = pieceCount-1;
        while(index>=0) {
              Box[] lineParts = subArray(pieces, pointer[index], index);
              SequenceBox line = new SequenceBox(lineParts);
              lines.addFirst(line);
              index = pointer[index]-1;
        }

        // finally: arrange the lines nicely inside a MultilineBox object.
        Box[] linesArray = lines.toArray(new Box[0]);
        MultilineBox result = new MultilineBox(linesArray,
              /* and other arguments */);

        return result;
}
```

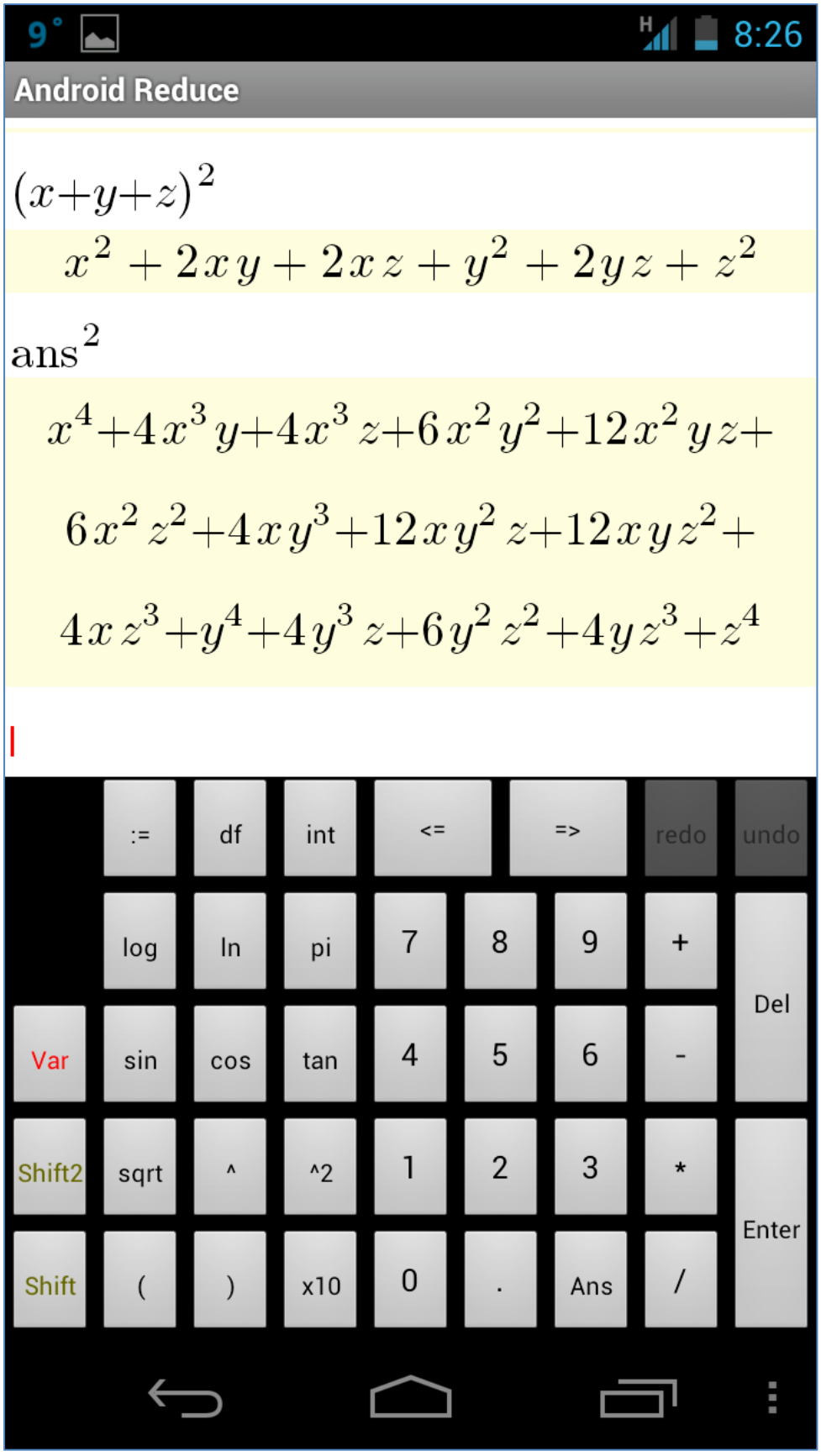
Appendix C App screenshots



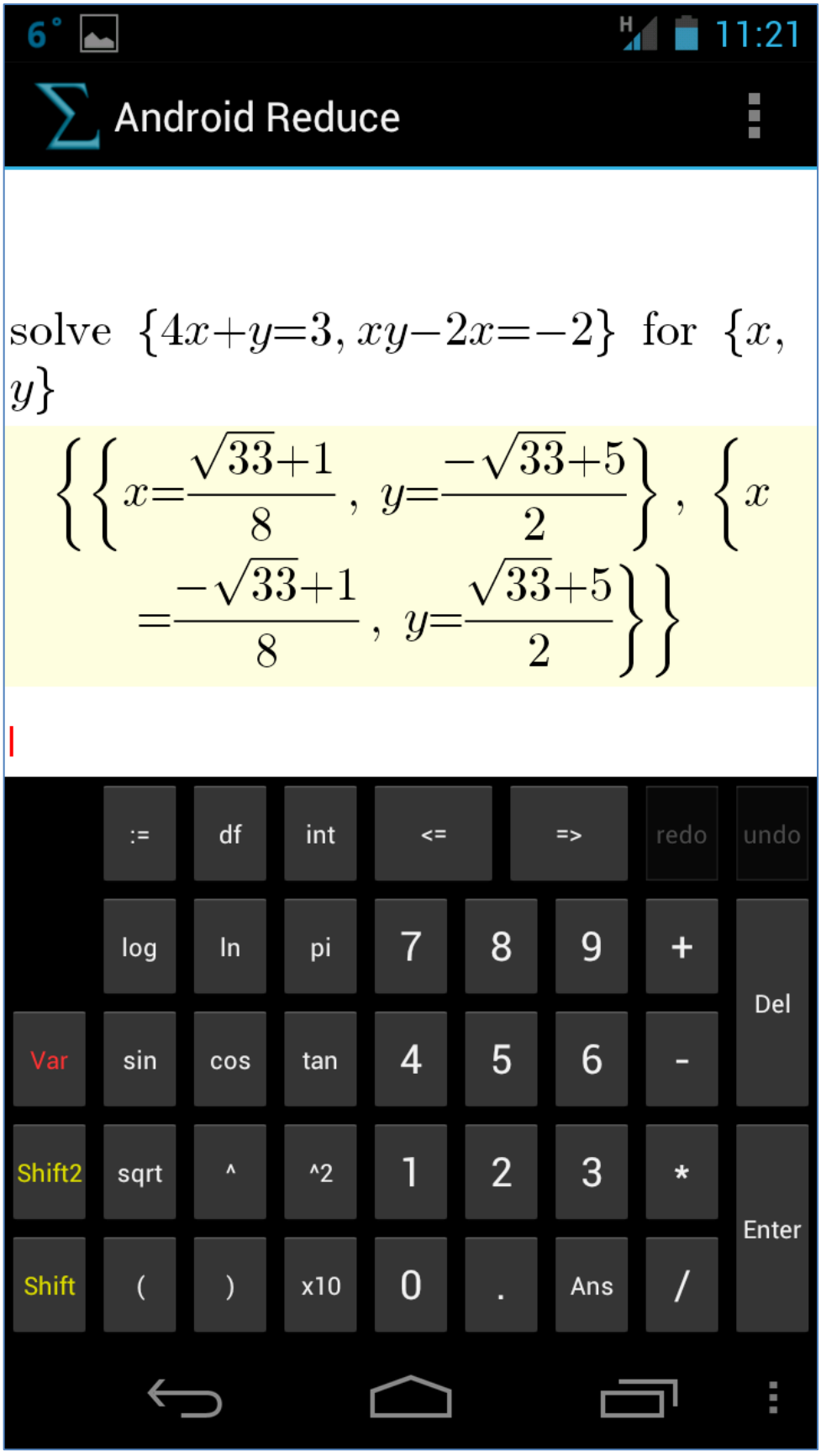
The application splash screen

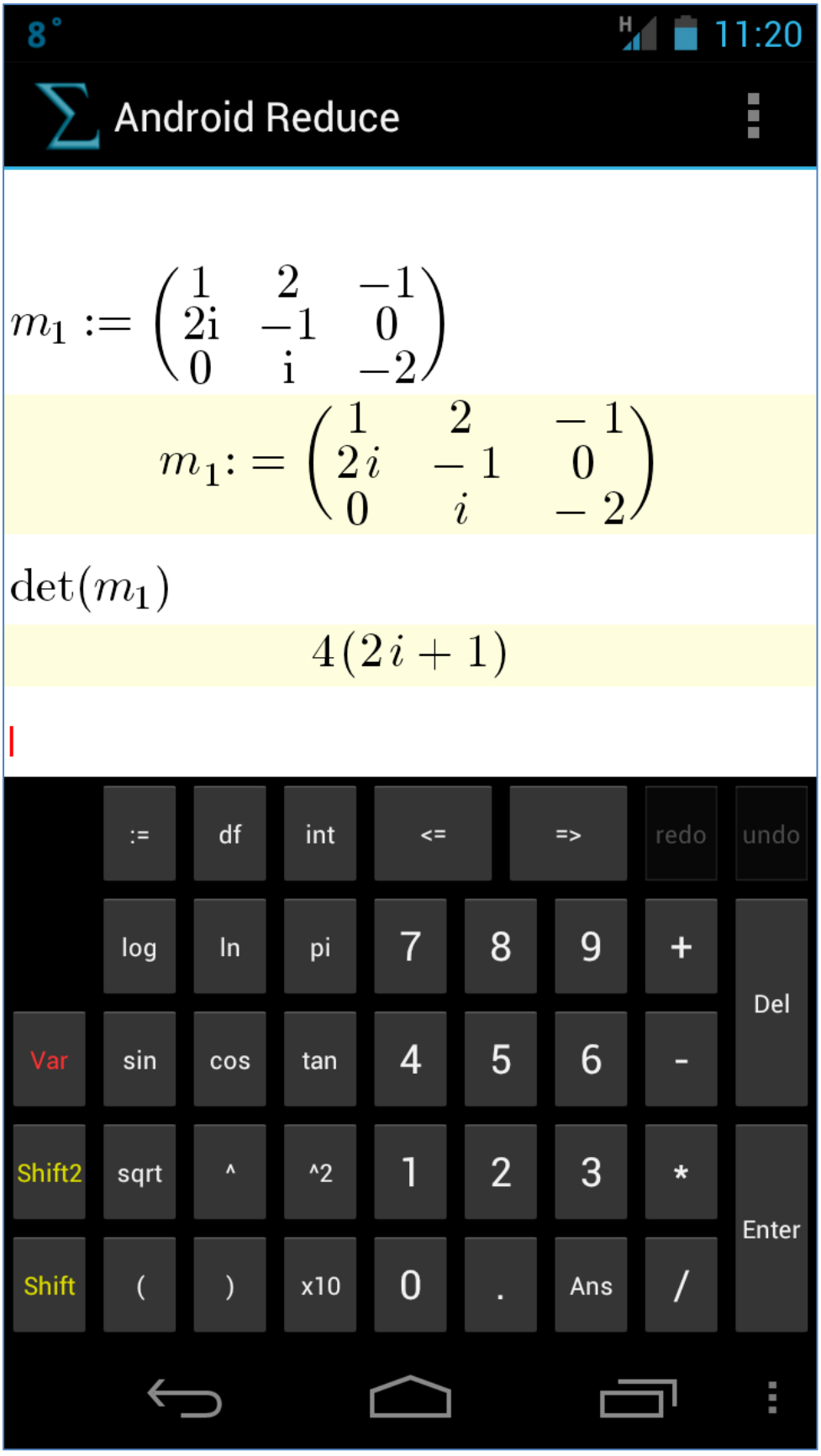


This snapshot demonstrates the importance of the formula fitting algorithm



The application interface running on versions of Android older than 4.0


6°
11:21
Android Reduce
solve {4x+y=3, xy−2x=−2} for {x, y}
{{x=(√33+1)/8, y=(−√33+5)/2}, {x=(−√33+1)/8, y=(√33+5)/2}}
:=
df
int
<=
=>
redo
undo
log
ln
pi
7
8
9
+
Del
Var
sin
cos
tan
4
5
6
-
Shift2
sqrt
^
^2
1
2
3
*
Enter
Shift
(
)
x10
0
.
Ans
/


8°
11:20
Android Reduce
$m_1 := \begin{pmatrix} 1 & 2 & -1 \\ 2\mathrm{i} & -1 & 0 \\ 0 & \mathrm{i} & -2 \end{pmatrix}$
$m_1 := \begin{pmatrix} 1 & 2 & -1 \\ 2i & -1 & 0 \\ 0 & i & -2 \end{pmatrix}$
$\det(m_1)$
$4(2i+1)$
:= df int <= => redo undo
log ln pi 7 8 9 + Del
Var sin cos tan 4 5 6 -
Shift2 sqrt ^ ^2 1 2 3 * Enter
Shift ( ) x10 0 . Ans /

Ahmad Akra
Trinity College
aa582

Computer Science Part II Project Proposal

Computer Algebra System for Phones and Tablets

21-Oct-2011

Project Supervisor:
Dr. Arthur Norman

Director of Studies:
Dr. Arthur Norman

Overseers:
Prof. Frank Stajano + Dr. Robert Mullins

Resources Form:
Submitted with this proposal

Introduction

A scan of the android app store reveals a variety of apps that perform algebra calculations. But most of them have limitations in the form they accept input, or the way they display output. And all of them have very little capabilities in the level of algebra complexity they can manage. Despite how powerful modern smart phones have become, algebra systems available for PC such as Mathematica or Maple still offer a lot more than similar systems available for Android.

The idea of this project is to bring the power of Reduce - an open source CAS written in Lisp and popular amongst mathematicians and scientists - to the Android platform, build an extensible and user friendly UI that interfaces to Reduce's functionality main core and display the formulae in a form closer to the real mathematical notation instead of text. I would like here to mention MathBot for the iPad as a feasibility demonstration.

There is a long history of interest in this sort of work, and the 1971 article by William Martin (Computer input/output of mathematical expressions) is perhaps where it all started. Regarding the formulae-on-small-screen problem, the TI algebraic calculators provide a reference point and background. The work done in this project will be evaluated both in terms of the extent to which it actually works and by comparing its capabilities with existing PC and Android software that works in the same general area, see the success criterion section for details.

Resources

As will be demonstrated in the next paragraph, most resources required for this project can be arranged without intervention from the computer lab.

This project is intended to run on the Android platform. And, as usual, all development will be on a computer running the Android SDK and Eclipse with the ADT plug-in (all which is free software). I will use my laptop for development. The majority of testing will be done on an Android emulator also running on the same computer. One such emulator is provided by the ADT plug-in mentioned above. I will occasionally need a real Android device for demonstration and deployment testing but That is not expected to be a problem since I know many people in our class who are happy to run the project on their Android phones and tablets once it is stable. Dr. Norman has an Android tablet and he said that he will be happy to test my project on it.

From the computer lab, I will only require an extra 2GB of space on the PWF to backup the Reduce code. This is included in resources form which was submitted alongside this proposal.

Starting Point

My project supervisor Dr. Norman has some existing Java code (~27K lines) which is an interpreter on top of Reduce[1], however it has not been written with the constraints of Android in mind. At present it provides a strictly text-mode interface where users type in commands and the results are displayed as "ASCII art" with fixed-width characters on the screen to give an approximation to mathematical notation.

I am familiar with the Java language, but completely new to Android development. I will be spending some time at the beginning to learn the Android mindset and development tools.

Project Structure

The project will be composed of

1 - A library for algebra manipulations. Dr. Norman's code will be used for this after it is ported to the Android platform. His code gives access to many of the features provided by Reduce CAS, such as solving polynomial equations and matrix multiplication.

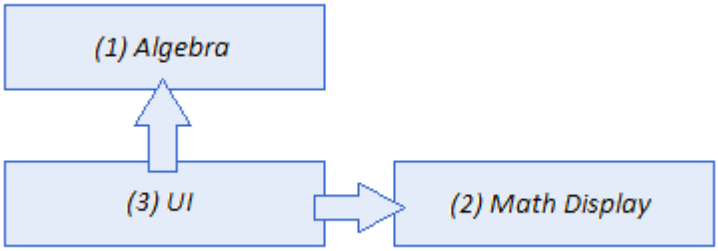
2 - An extensible library for mathematical typography display. This should allow a formula like:

(1/2) * sqrt(x^2 - 1)

to be displayed closer to the real mathematical notation:

$$\frac{1}{2}\sqrt{x^2-1}$$

3 - An Android UI that interfaces with the previous two libraries.



[1] The source code for the Java interpreter and all the associated Reduce Lisp files are available on SourceForge and can be acquired by running the following shell command:
`svn co http://reduce-algebra.svn.sourceforge.net/svnroot/reduce-algebra reduce-algebra`

Success Criterion

This project is expected to be challenging, many of its challenges result from the nature of smart phones and tablets and the fact that they have small screens that work by touch, some of these challenges appear to not have been seriously addressed before, The main success criteria for this project will be:

A - Porting Dr. Norman's code to Android. This will demonstrate an important programming skill which is the ability to read, modify and reuse existing code written by other programmers. Amongst other things, this will require mastery of the Java programming language.

B - Arranging that the output is displayed in a form that is closer to genuine mathematical typography than plain text mode. This will demonstrate problem solving skills when it comes to layouting the formulae on the small screen.

C - Building an extensible and user friendly input interface for Android devices. This will demonstrate several skills:

1. The ability to quickly grasp a new development platform (I am new to Android development).
2. The ability to structure the project and follow the best programming practices that allow extensibility and testability of an Android UI project.
3. And the ability to construct the interface such that it is usable and comfortable to look at.

Within the scope of a Part II project it is not likely to be feasible to complete a careful assessment of the HCI issues involved, and so the code implemented will be built with flexibility in mind as a technology demonstration and prototype. There will obviously be scope for many extensions, added features and refinements - for instance: scrolling, zooming, editing of existing input and export of results. But the key success criteria will be the ability to demonstrate success against the three objectives listed above.

As part of the evaluation, the outcome of this project will be compared to existing software in the Android app store:
(1) Algebra Calculator Pro from MJH Mobile DEV.
(2) Algebraic Calculator by Tarcin.
(3) MathScript Calculator from The Think Tanks
In particular, apps 1 and 2 will be compared to this project in terms of how many different algebra problems they can solve. App 3 is a very powerful algebra app and will be compared to this project in terms of the convenience of its input system and the elegance of its displayed output.

Schedule and Milestones

21st October - 15th November

1 - Become familiar with the Android development mindset and tools.
2 - Establish a backup plan.
3 - Research and identify some existing CAS software that will be used as a reference to improve and compare against during development.

Milestone: Create simple Android project that takes textual input and produces textual output and run it on a real Android device.

16th November - 31st December

1 - Obtain and install Dr. Norman's algebra library on my laptop.
2 - Understand the library and identify regions of the code that will need rewriting to run on the Android platform.
3 - Move the code to an Android project, and run it.

Milestone: Create a very simple interface that takes textual input through a dialog box and displays textual output.

1st January - 29th February

Temporarily ignore the algebra library. And develop an extensible math display library that can display a basic range of formulae in mathematical notation. An example to illustrate the problems that will be tackled here is the problem of where to break the formulae when it doesn't fit on a small screen.

Milestone: Use the math display library to display (on an Android screen) a range of simple formulae constructed by hand.

1st March - 7th March

Milestone: Use the math display library to display simple output generated from Dr. Norman's algebra library.

8th March - 15th April

1 - Finalize the general UI design of the application
2 - Finalize the design of the input system which is a custom keyboard of symbols and operators. These will be organized in a tree structure such that only the most commonly used symbols and operators are visible and the rest can be reached through sub-menus. The system keyboard is only used for inputting multi-character names.
3 - Build the UI on the Android platform.

Milestone: Connect the input system to the Algebra library and arrange the UI to display formulae using the math display library created earlier. At this point the project should be in the beta stage.

16th April - 30th April

1 - Review the code for the last time.

2 - Perform final debugging and testing.

1st May - 3rd May

Evaluate the project.

4th May - 18th May

Write and submit the dissertation.