# System Requirements and System plan

The K-Mapper Project V2.0

N. House, K. Levner, J. Pledger, J. Ranger

February 2nd, 2010

**Abstract**

The goal of this project is to create a tool to allow a user to draw Karnaugh maps of up to 6 variables. Input can be accepted as an equation, truth table, Karnaugh map, or imported via a file (in pla form). The application will minimize the K-Map to a minimal sum of products and product of sums form. The user will have the option to modify their input (even in a different input method than originally used) after obtaining output and the user may export their results to a file (in pla form) as well.

This application is designed with the intended audience of Computer Science 2610 (Digital Systems) students. They are expected to have a background in KMaps, Truth Tables, and Boolean logic equations.

It is intended to be run on the Linux machines found in the U of L computer labs. To be built, this tool will make use of Qt, a GUI programming library compatible with C++. It will be documented using Doxygen, a popular documentation tool. Both of these libraries are open source and come at no charge.

The design in this document has been created with the expectation of future scalability concerning the number of variables, additional widget modification and additional input and output types. The design primarily focuses on encapsulation, which provides abstraction between the front and back end, but can still be linked using the adaptor design pattern. This is intended to give the user a great experience while still logically organizing data in the background.

## Version History

| *Version* | *Date* | *Author(s)* | *Changes* |
|---|---|---|---|
| 1.0 | 2/2/2010 | NH, CB, JB, BK | • Initial version |
| 1.1 | 2/9/2010 | NH, CB, JB, BK | • Minor grammatical corrections |
| 2.0 | 3/17/2010 | NH, JR, KL, JP | • Updated object models to reflect refactoring (See section 3.1)<br>• Updated program screenshots<br>• Added additional coding conventions<br>• Fixed additional grammatical errors |
| 2.1 | 3/23/2010 | NH, JR, KL, JP | • Added legends to activity model and object model<br>• Updated equation input dialog screenshot<br>• Fixed more grammatical errors |

# Contents

## 1 Introduction

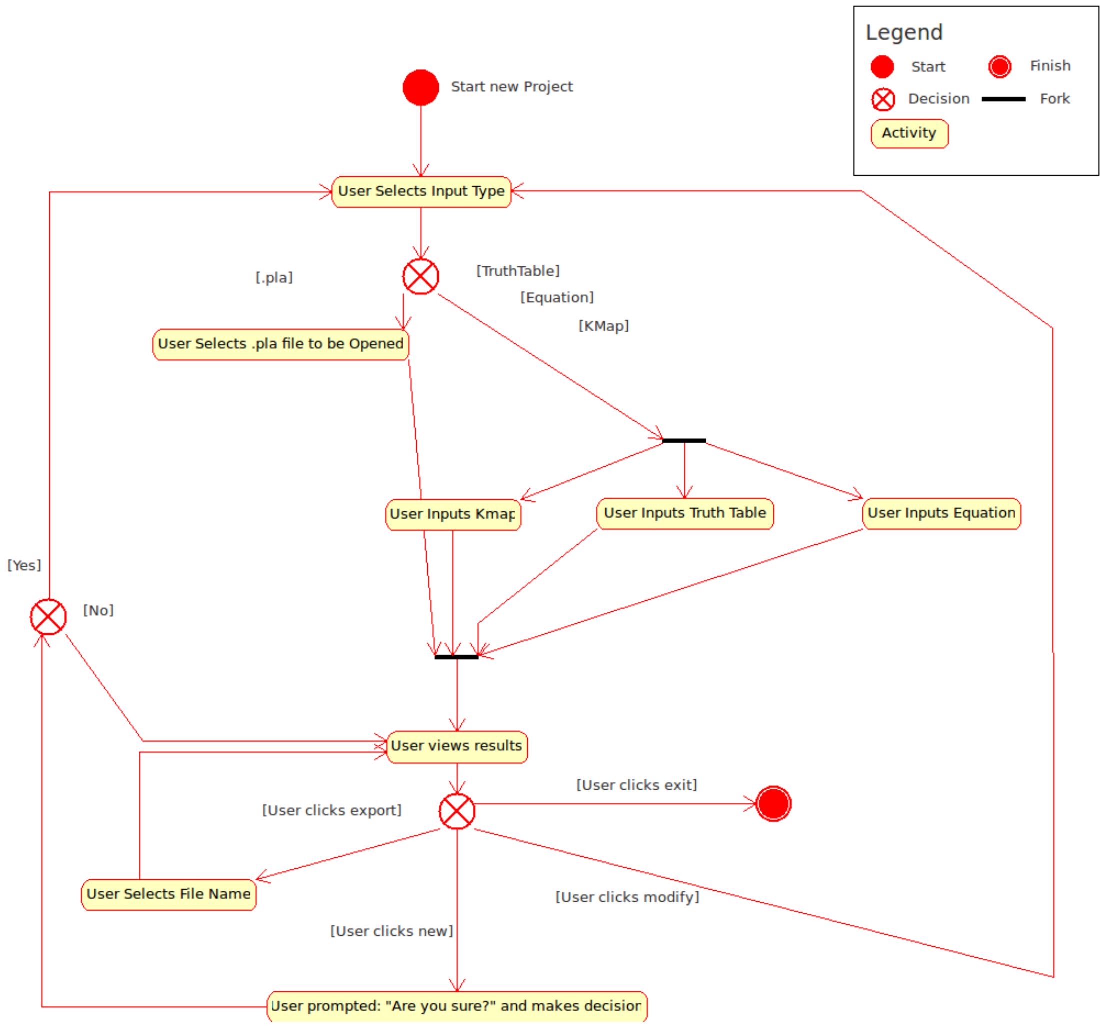
This document intends to serve as an outline of the implementation of the KMapper project in it's entirety.

## 2 Behavioral Models

### 2.1 Activity Model

This model (fig. 1) represents the data flow corresponding to the choices that the user makes. First, the user will be prompted to select their input type. If the user chooses to input a truth table, Equation or Karnaugh map, they will be prompted to select how many variables they wish to use. After submitting their input or importing a file, the user will view the results. The user will then have the ability to export the data to a file, modify the input, start new input, or quit.

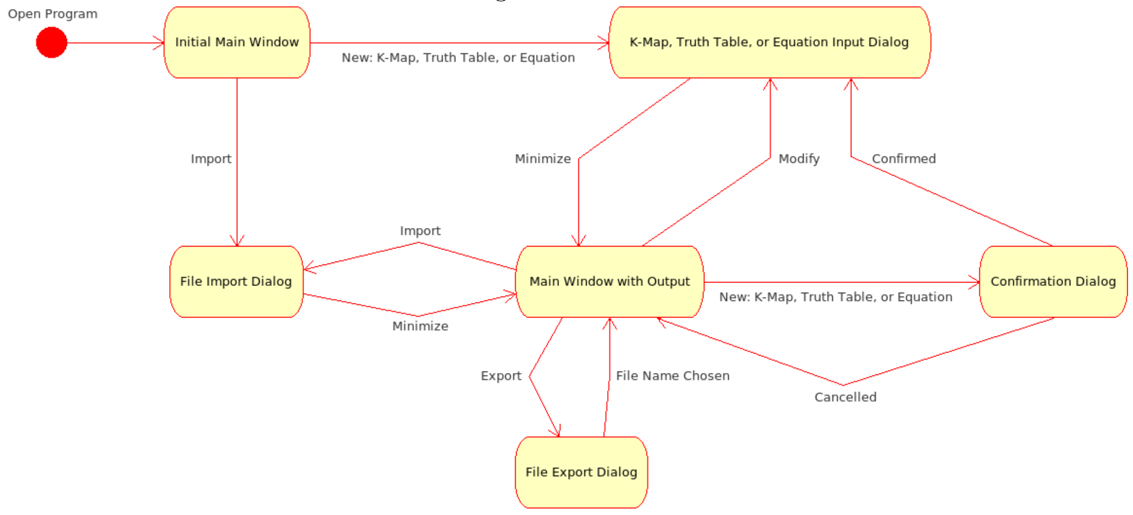
Figure 1: Activity Diagram



## 2.2 State Model

The state model in fig. 2 shows the different states of the program's GUI from the user's point of view. These specific states represent points in the program in which the user will have to choose an action in order to move to another state. From any state, the user may also choose to exit the program.

Figure 2: GUI State Model



# 3 Object Models

The object model presented here outlines the classes to be implemented in the project, as well as some of the required inheritance from external libraries.

In this model, the adaptor classes provide methods to the GUI front end to interpret data from DataBackend into the appropriate GUI element. It also allows communication in the opposite direction, where elements in the graphical front end can save their data into the back end. All of the adaptor classes are pure virtual, that is, their functions must be implemented by the DataBackend. This ensures compatibility of any future Backend representation with the current front end.

In addition to the adaptor classes, there are two widget classes that have been created to provide a specialized way of displaying both truth tables and K-Maps. These widgets will be created by either creating a specialization of the QTable widget, or by implementing a new widget by sub-classing QWidget. A specialized widget for inputting equations will probably not be necessary, as the QTextBox widget provides sufficient functionality to input a string.

The final type of class in the model is the Dialogs, which are sub-classed from QDialog. These are specialized dialogs used to either input new data to the back end, or modify the existing data. They will contain the appropriate type of widget, depending on the type of input the dialog is meant to take. It should be noted here that a specialized dialog for file input is unnecessary, as the provided QFileDialog is sufficient.

More detailed descriptions of the individual classes can be found in section 4.

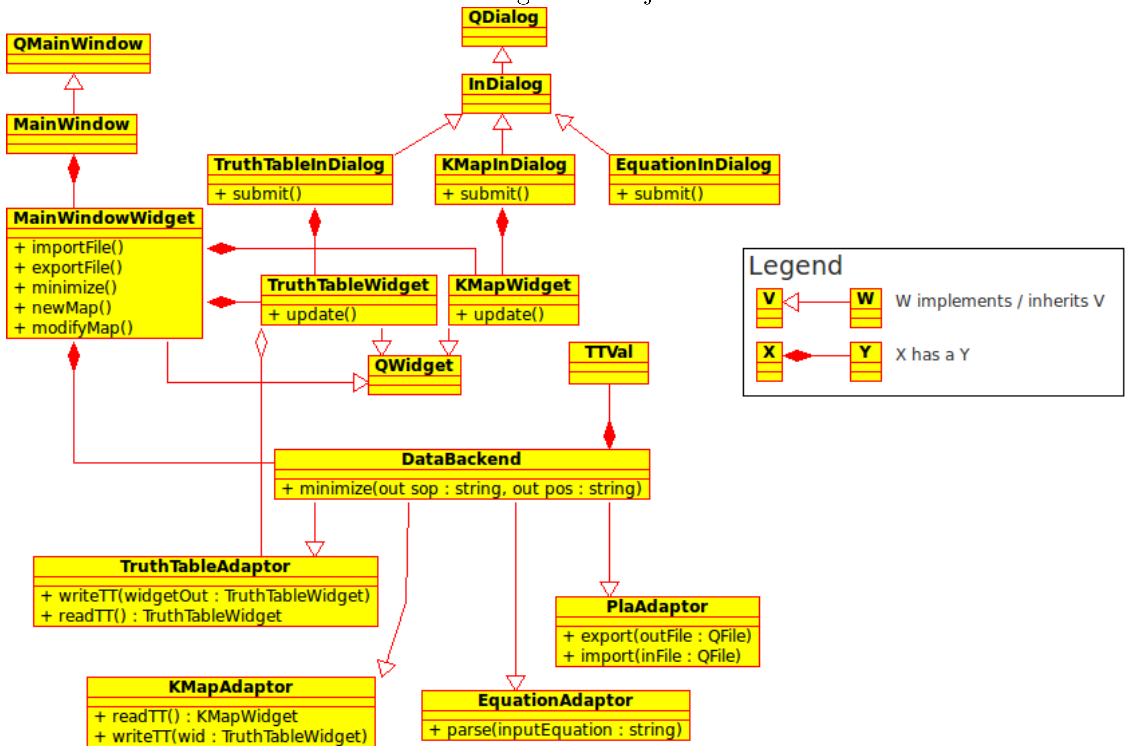
## 3.1 Changes In Version 2.0

Since the first iteration of the project, the following has changed in our object model.

- TTVal class has been added to wrap the enumerated type which is the underlying data type of the DataBackend

- Adaptor classes are now pure virtual classes, as they can only declare the interface that must be implemented in the DataBackend. This is standard for the adaptor pattern.
- DataBackend now inherits from all of the adaptor classes, as it must implement that functionality to ensure proper function of the front end
- Added MainWindowWidget, a class which acts as the central widget of the Main Window
- Added InDialog class, a super class which includes the common functionality of all the input dialogs

Figure 3: Object Model



## 4 Class Descriptions

These class descriptions are intended to give a introduction to the functionality of each class, and an overview of how the classes interact with each other. For a graphical representation of the object model, see figure 3.

### 4.1 DataBackend

The DataBackend class holds all the data and implements the Quine-McCluskey algorithm to reduce the truth table to minimal POS and SOP form. This class is intended to house the entirety of back end functionality, as well as to implement any functions inherited from the adaptor classes.

### 4.2 EquationAdaptor

The EquationAdaptor class is a pure virtual class which is inherited by DataBackend. It provides declares a conversion method called parse() which accepts a string, and then parses it into the common data format inherited from DataBackend.

### 4.3 EquationInDialog

EquationInDialog provides a dialog which allows users to input a truth table in equation form. This class uses methods from EquationAdaptor to pass data into the back end.

### 4.4 InDialog

Super class to all the other input dialogs, which provides methods/widgets to change number of variables and variable names.

### 4.5 KMapAdaptor

The KMapAdaptor class is a pure virtual class which is inherited by DataBackend. It provides the interface necessary for the KMapWidget to display the data.

### 4.6 KMapInDialog

The KMapInDialog provides and input dialog which allows the user to either input a new K-Map or modify the one currently stored in the back end, as well as change the number of variables and variable names. User interaction with data occurs via a KMapWidget, which in turn writes to the back end via functions defined in KMapAdaptor.

### 4.7 KMapWidget

The KMapWidget is a graphical representation of a K-Map, and is a specialization of Qt's QTableWidget. QTableWidget already implements most of the functionality required, the major changes are for the style and input methods of the widget.

### 4.8 MainWindow

The MainWindow class contains the applications main interface. It inherits all functionality from QMainWindow, as well as providing all the useful data display methods. It contains a DataBackend object, which is the central data location for the entire program. A graphical presentation of the main window can be found in section 5.

### 4.9 MainWindowWidget

The MainWindowWidget acts as the central widget to the MainWindow, and holds all widgets to be displayed on the MainWindow.

### 4.10 PlaAdaptor

PlaAdaptor is another pure virtual adaptor class inherited by DataBackend It provides file input and output through the following two functions:

- import(QFile*) - reads file specified by a QFile object and stores the info in the Backend
- export(QFile*) - writes the content of the Backend to file specified by a QFile object

This class has nothing to do with actually getting the file name from the user, it simply opens a preselected file (possibly coming from Qt's built in FileDialog) and either reads or writes to it.

### 4.11 TruthTableAdaptor

The TruthTableAdaptor class is a pure virtual base class inherited by DataBackend. It provides function definitions for all accessor methods needed for the TruthTableWidget

### 4.12 TruthTableInDialog

The TruthTableInDialog provides and input dialog which allows the user to either input a new truth table or modify the one currently stored in the back end, as well as modify the number of variables and the variable names. User interaction with the data occurs via a TruthTableWidget, which in turn writes to the back end via functions defined in TruthTableAdaptor.

### 4.13 TruthTableWidget

The TruthTableWidget is a graphical representation of a truth table, and is a specialization of Qt's QTableWidget. QTableWidget already implements most of the functionality required, the major changes are for the style and input methods of the widget.

### 4.14 TTVal

TTVal is a class which contains the data type for each element in the Truth Table. This wrapper provides the accessor and modifier methods needed to read, write, and minimize the truth table, such as conversion to string, incrementing by one, and comparison.

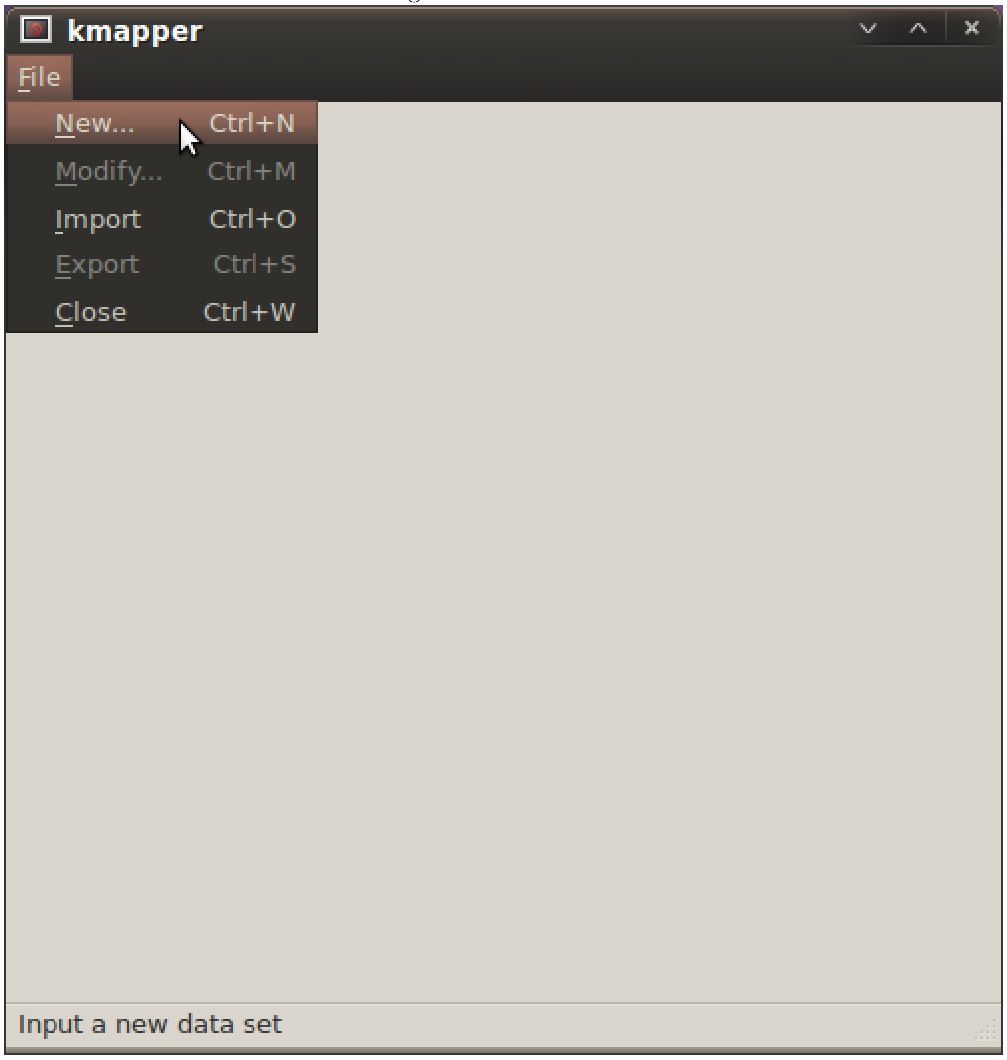
## 5 UI Design

### 5.1 Inputting New Data

To input new data to the program, in any format, the user uses the "File → New" menu (fig. 4). It presents the three possible formats (figure 5), and each selection opens it's own specific input dialog (figures 6, 7, and 8).

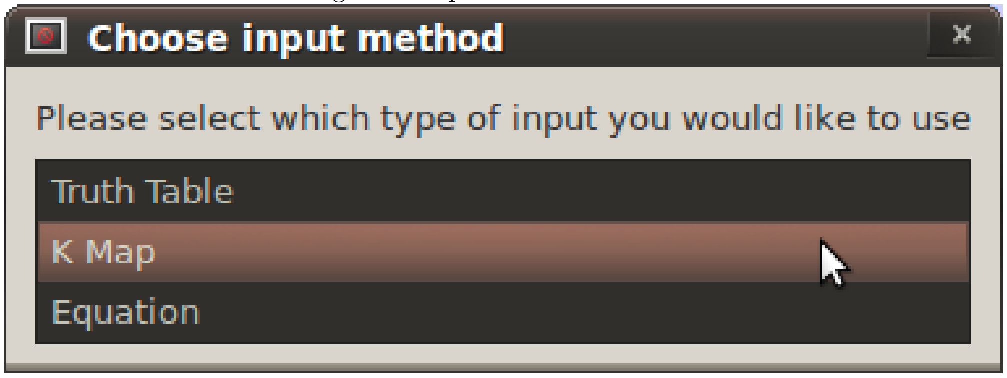
The input dialogs allow the user to select the number of variables, the variable names, and finally input the content of the table. Clicking "Ok" closes the dialog and show the updated main window.

Figure 4: New Menu



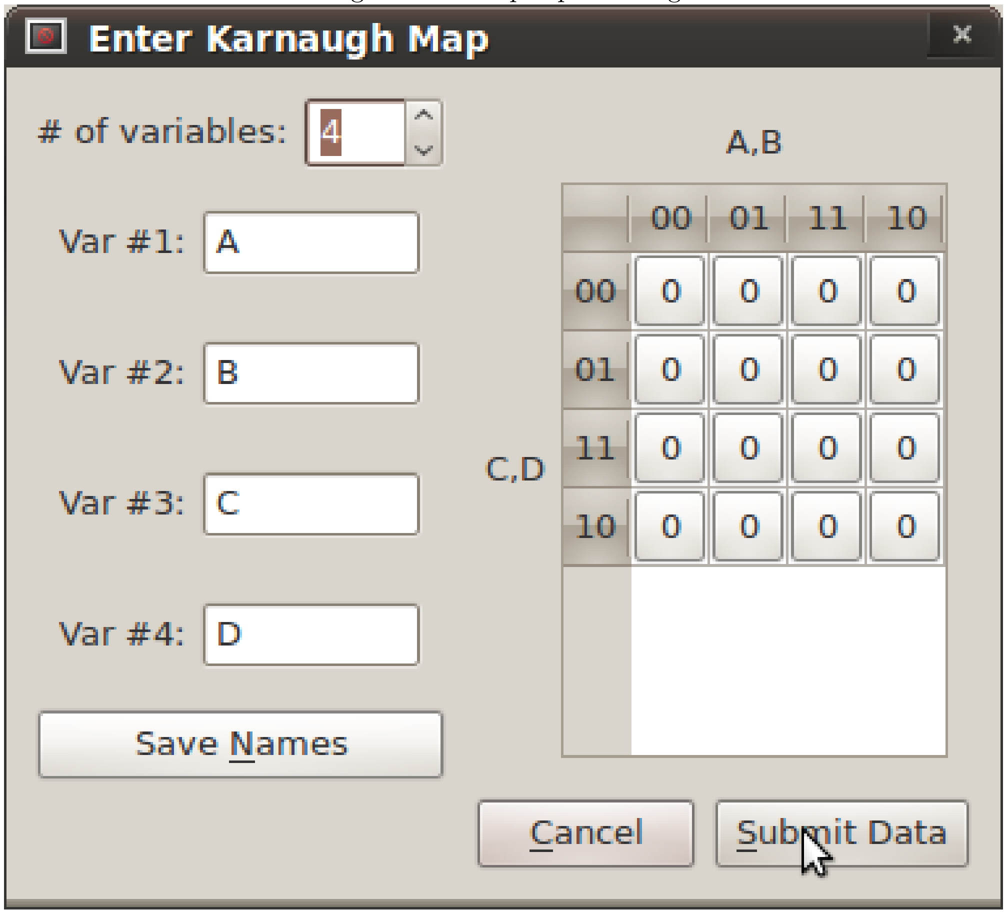
Input a new data set

Figure 5: Input Method Selection



Please select which type of input you would like to use

Figure 6: K-Map Input Dialog



Enter Karnaugh Map

# of variables: 4

Var #1: A

Var #2: B

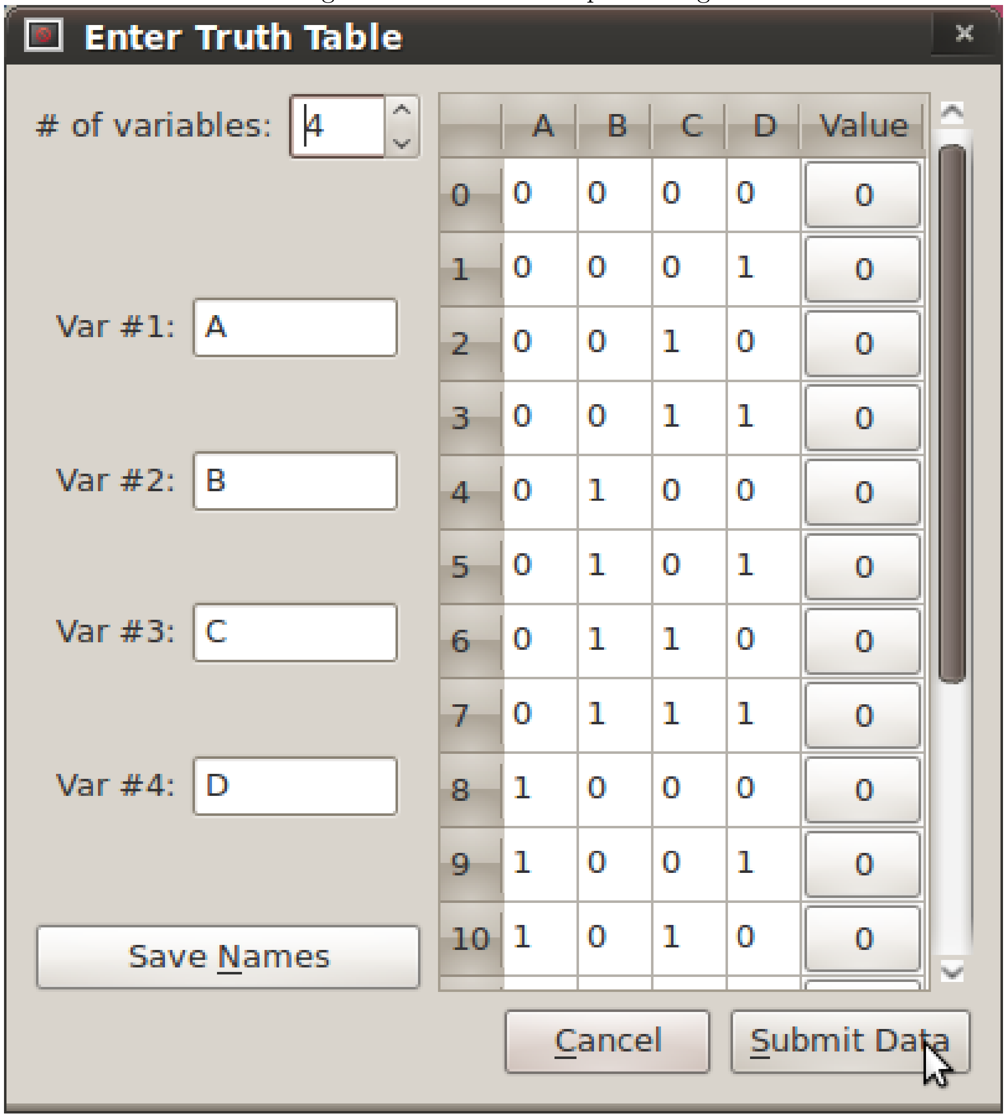
Var #3: C

Var #4: D

A,B

| C,D | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 |
| 01 | 0 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 |

Figure 7: Truth Table Input Dialog



# of variables: 4

Var #1: A

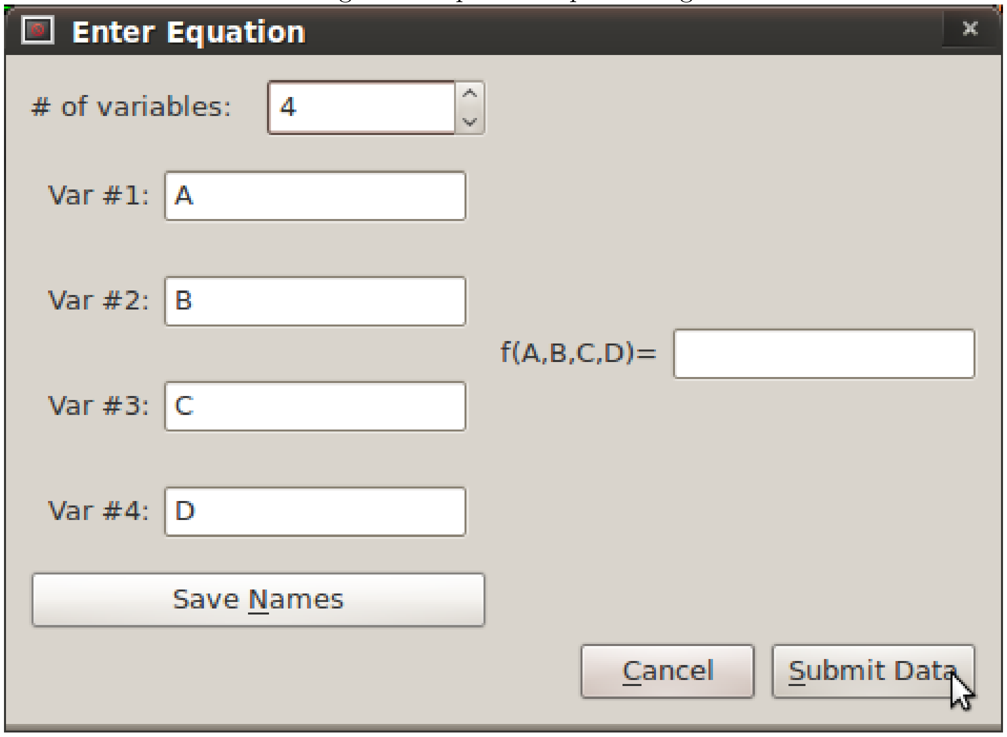
Var #2: B

Var #3: C

Var #4: D

| | A | B | C | D | Value |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 0 |
| 8 | 1 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 0 |
| 10 | 1 | 0 | 1 | 0 | 0 |

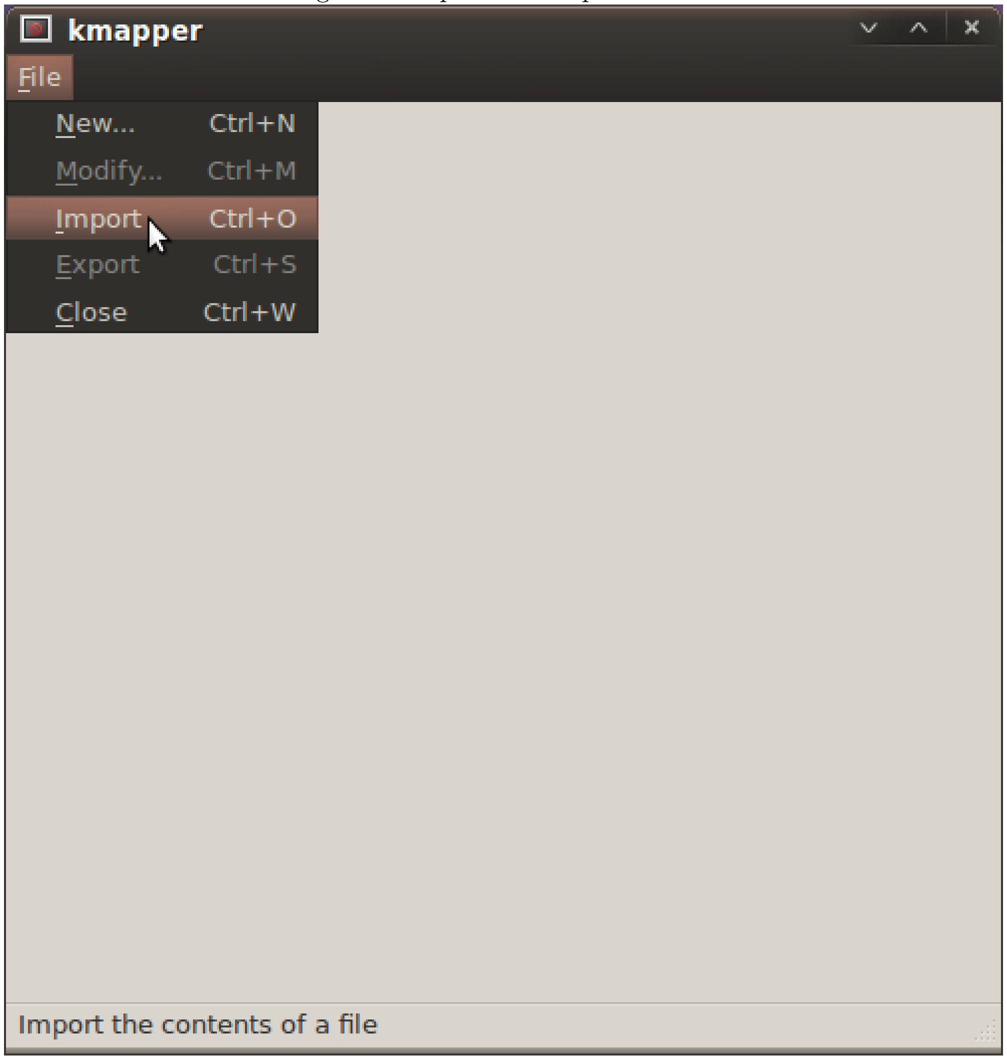
Figure 8: Equation Input Dialog



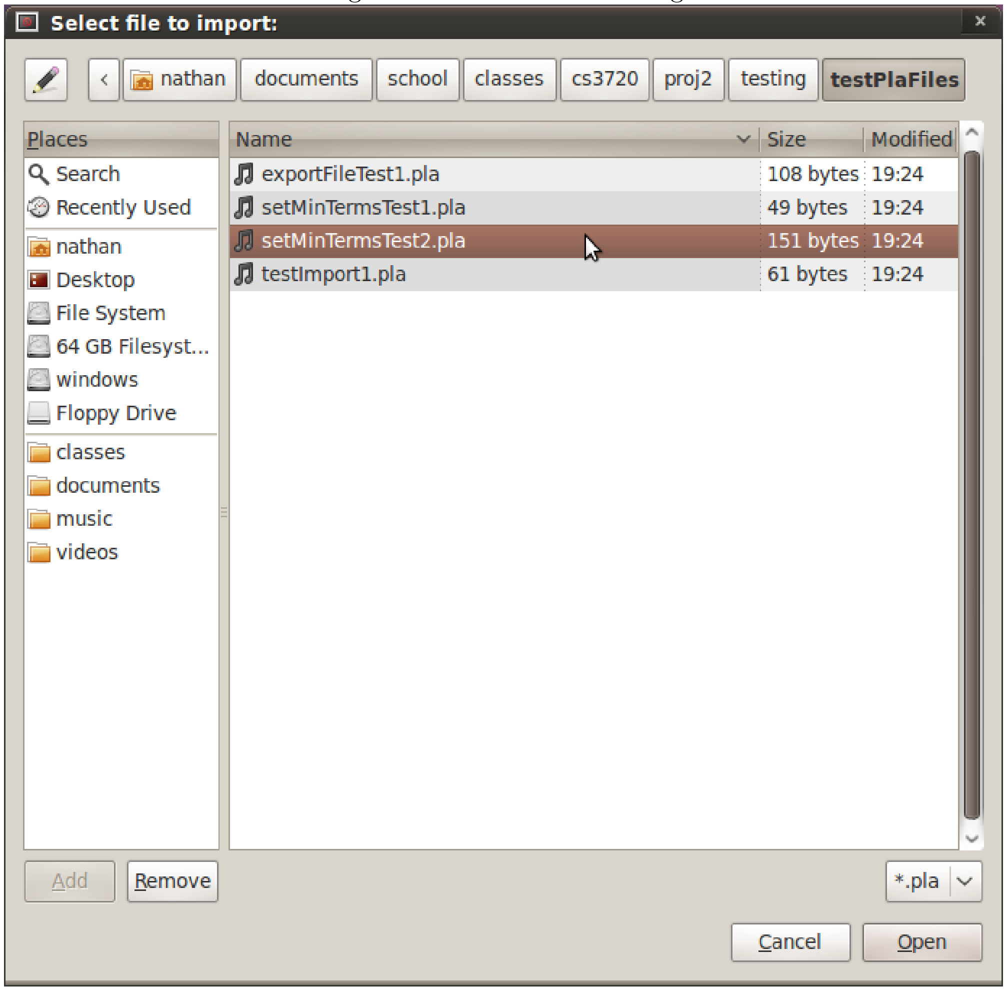
## 5.2 File Operations

Importing and exporting data to/from file is also accomplished via the “File” menu (fig. 9). Upon clicking import, the user is presented with a file chooser dialog (fig. 10). After that the primary window will be presented with the appropriate information loaded and the results will be viewable (See section 5.3). If the user clicks export, the same file chooser dialog will be presented, and the current data will be saved to the file. If the save operation fails, an appropriate message will be presented to the user.

Figure 9: Import and Export Menus



Import the contents of a file



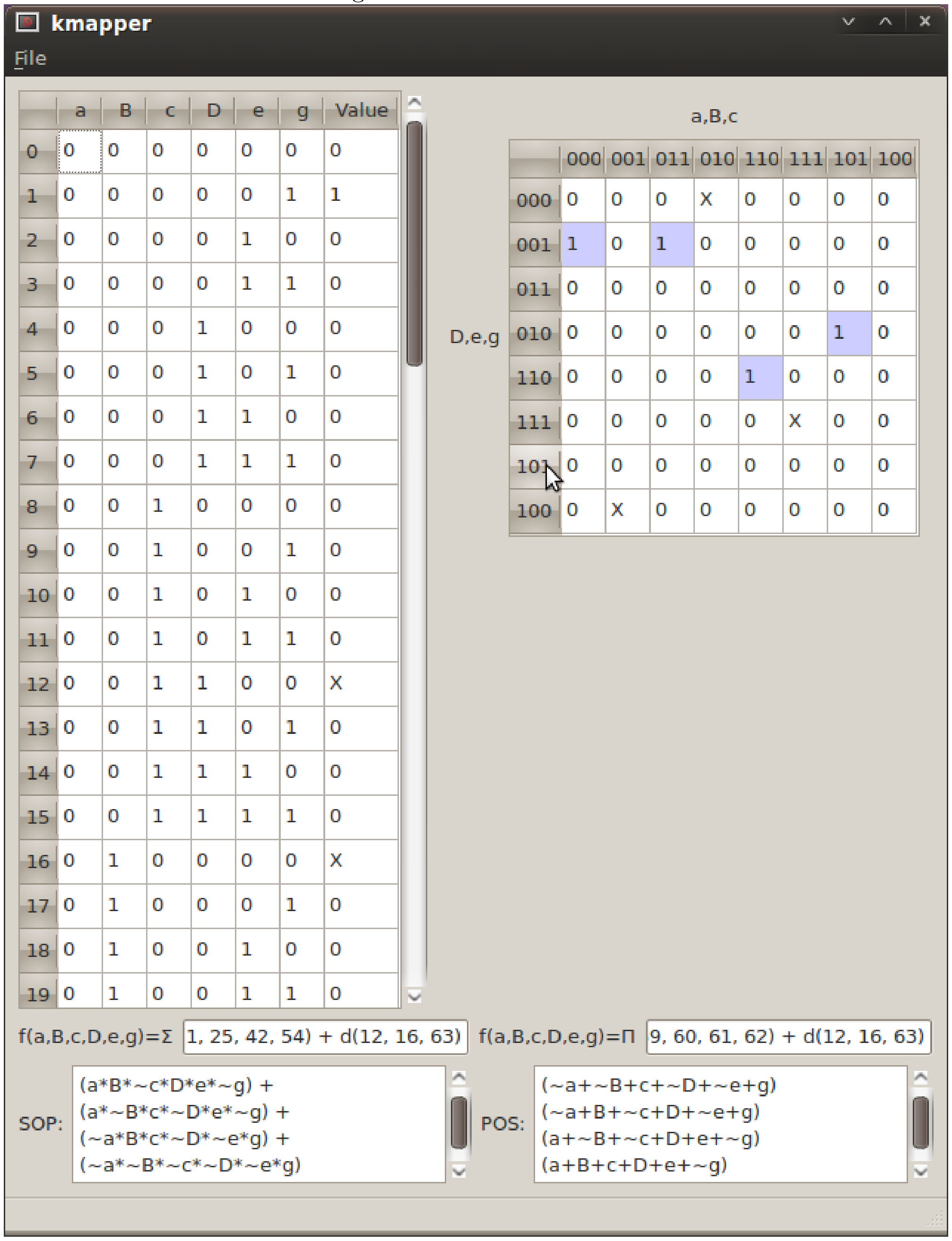
| Name | Size | Modified |
|---|---|---|
| exportFileTest1.pla | 108 bytes | 19:24 |
| setMinTermsTest1.pla | 49 bytes | 19:24 |
| setMinTermsTest2.pla | 151 bytes | 19:24 |
| testImport1.pla | 61 bytes | 19:24 |

Figure 10: File Chooser Dialog

## 5.3 Viewing Results

Figure 11 show the results of minimization. From there the user can modify via "File → Modify" (see fig. 12), or export to file via "File → Export"

Figure 11: View of Results



kmapper

File

| | a | B | c | D | e | g | Value |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 4 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 6 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 8 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 10 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 11 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 12 | 0 | 0 | 1 | 1 | 0 | 0 | X |
| 13 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 14 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 15 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 16 | 0 | 1 | 0 | 0 | 0 | 0 | X |
| 17 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 18 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 19 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |

a,B,c

D,e,g

| | 000 | 001 | 011 | 010 | 110 | 111 | 101 | 100 |
|---|---|---|---|---|---|---|---|---|
| 000 | 0 | 0 | 0 | X | 0 | 0 | 0 | 0 |
| 001 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 011 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 010 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 110 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 111 | 0 | 0 | 0 | 0 | 0 | X | 0 | 0 |
| 101 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 100 | 0 | X | 0 | 0 | 0 | 0 | 0 | 0 |

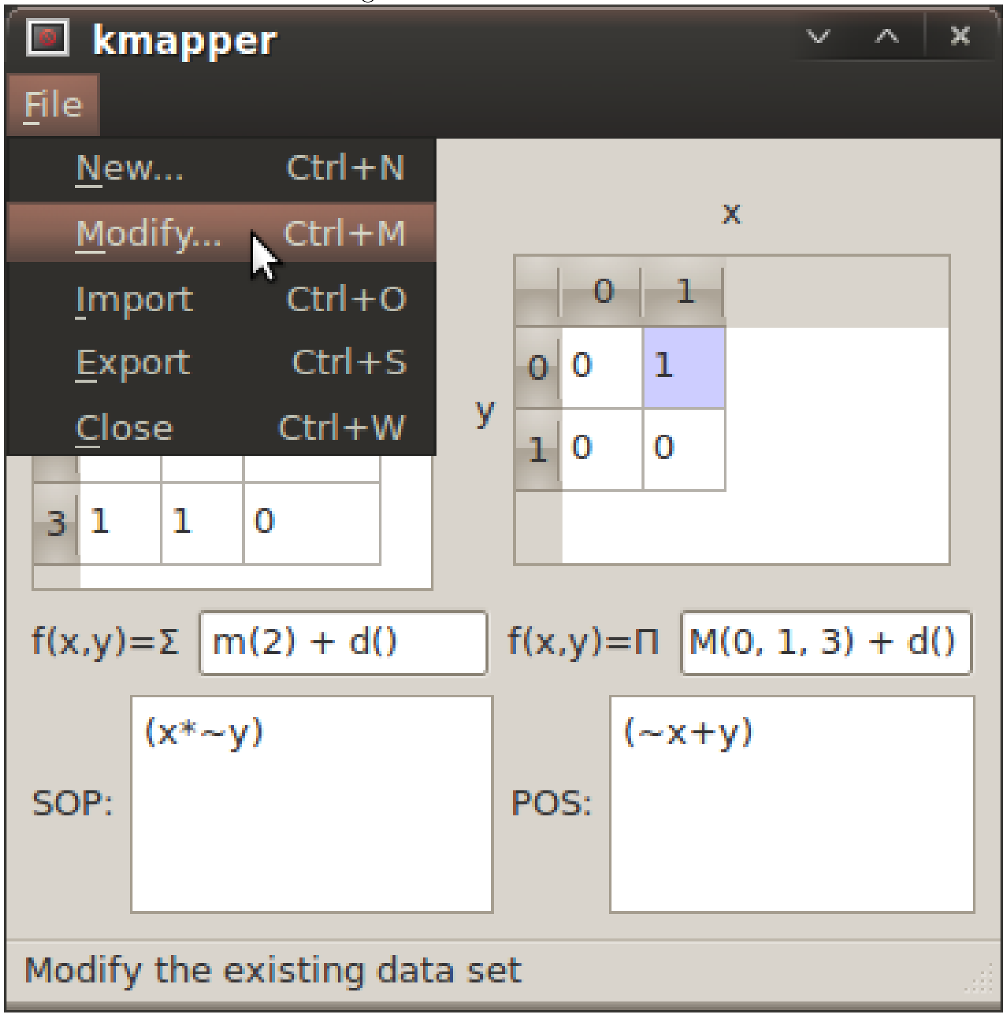
f(a,B,c,D,e,g)=Σ 1, 25, 42, 54) + d(12, 16, 63)

f(a,B,c,D,e,g)=Π 9, 60, 61, 62) + d(12, 16, 63)

SOP:
```
(a*B*~c*D*e*~g) +
(a*~B*c*~D*e*~g) +
(~a*B*c*~D*~e*g) +
(~a*~B*~c*~D*~e*g)
```

POS:
```
(~a+~B+c+~D+~e+g)
(~a+B+~c+D+~e+g)
(a+~B+~c+D+e+~g)
(a+B+c+D+e+~g)
```

Figure 12: View of Results 2



|  | x=0 | x=1 |
|---|---|---|
| y=0 | 0 | 1 |
| y=1 | 0 | 0 |

## 6 Coding Standards

### 6.1 Variables - Naming

- Global variables and enumerated types will be name in all capitals, with words separated by underscores.
  - **const int** MAX_NUMBER_OF_VARS=6;
  - **enum** E_TYPE{NUM=256, GRP=257};
- Class and Struct data types will start with a capital letter, and capitalize each new word.
  - **class** ClassExample{ ... };
- All others will be named using standard camel casing(lower case to start followed by capitals for each new word).
  - **int** thisIsAnExample;
  - string myName;

## 6.2 Variables - Miscellaneous

- Global variables will always be declared constant.
- Enumerated types will have their values manually defined and start at a minimum value of 256 to prevent potential conflicts with ASCII characters.

## 6.3 Files

- File names should be indicative of what they contain and/or what they do.
- File names should not have spaces.
- Temporary files(for file I/O) shall be used whenever loss of data map occur.

## 6.4 Commenting

- Comments shall follow the Doxygen coding style. http://www.doxygen.nl/manual.html
- Comments for both functions and classes shall be placed directly above the item being described.
- Brief descriptions are in header files, and should briefly describe what the function is doing
- Long descriptions are in implementation files, and should briefly explain how the function will be completing it's goal
- Variables should be commented in line, with the ///< tag
- For a sample header file, see appendix A

## 6.5 General

- Code revisions should compile before being submitted, if they do not:
  - Comment out the section of code changed so it will compile.
  - Leave a message when committing(-m flag) stating what/where the error is coming from.
- Commit changes before starting a new section/branch of coding.
- If a function is returning 'placeholder data', make sure that it is obviously a placeholder value.
  - Using a -1 when returning an integer.
- Functions should be grouped in the header file in some logical way
- Functions in implementation files should be in the same order as they appear in the header file
- Sections of [repeated] similar code that can be converted to functions should be converted.

## A Sample Header

```
/**********************************************************************//**
\file
\brief Interface file for the Filename class

\date March 23rd, 2009
\author Nathan, Kate, Jason, Jacob
************************************************************************/


#ifndef FILENAME_H
#define FILENAME_H


/**********************************************************************//**
\brief One line description of the class

Longer explanation about the class and what it does or how it relates to
other classes.
************************************************************************/
class Filename{
 /// Constructor
 Filename();

 /// Destructor (if needed)
 ~Filename();
 /// What this function does
 int myFunction();
 ///What this function does.
 int area();

 private:
 int variableName1; ///< What this variable is
 int variableName2; ///< What this variable is
};


#endif
```